VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Triangulace ořezaných NURBS ploch

Tessellation of Trimmed NURBS Surfaces

2012 Ondřej Staňkovič

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Ondřej Staňkovič**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Triangulace ořezaných NURBS ploch
Tessellation of Trimmed NURBS Surfaces

Zásady pro vypracování:

Cílem práce je navrhnout a implementovat robustní metodu pro triangulaci ořezaných NURBS ploch včetně pokročilých funkcí jako např. svařování. Implementace by měla být provedena s ohledem na systém VRUT a jeho podporu NURBS entit.

1. Seznamte se s NURBS plochami a s jejich ořezáváním.
2. Prostudujte doporučenou literaturu a publikace týkající se triangulace NURBS.
3. Z prostudovaných materiálů vyberte nejvhodnější postup, případně ho vhodně rozšiřte.
4. Metodu implementujte (C++, zvažte možnosti paralelizace).
5. Výslednou funkčnost otestujte na dodaných modelech.

Seznam doporučené odborné literatury:

[1] PIEGL, Les; TILLER, Wayne. The NURBS Book. 1997. 646 s. ISBN 3-540-61545-8.
[2] BALÁZS, Ákos; GUTHE, Michael; KLEIN, Reinhard. Efficient trimmed NURBS tessellation. In Journal of WSCG 2004.
[3] KAHLESZ, F; GUTHE, Michael; KLEIN, Reinhard. NURBS rendering in OpenSG Plus. 2002.
[4] HJELLE, Oyvind. Triangulations and Applications. 2010. 234 s. ISBN 978-3642069888.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * FAKULTA elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 18. dubna 2012 .....

Abstrakt

Cílem mé diplomové práce bylo vytvořit dva moduly pro nástroj virtuální reality VRUT, vyvíjený a využívaný společností Škoda auto a.s.. Moduly slouží k načtení geometrických NURBS ploch uložených ve formátu IGES a k jejich následnému převodu na trojúhelníkovou síť. Tato práce popisuje jaké postupy a algoritmy byly při vývoji použity a s jakými problémy a komplikacemi jsem se musel vypořádat.

Klíčová slova: Tesselace, IGES, NURBS křivky a plochy, aproximace křivek a ploch, Delaunayho triangulace

Abstract

The aim of the master thesis was to make two modules for a tool of virtual reality VRUT, developed and used by Škoda auto a.s. Modules are used for loading geometrical NURBS surfaces saved in IGES format and for their subsequent transfer to triangular net. This thesis also describes which working procedures and algorithms were used during the development and which problems and complications I had to deal with.

Keywords: Tessellation, IGES, NURBS curves and surfaces, curve and surface approximation Delaunay triangulation

Seznam použitých zkratek a symbolů

CAD	–	Computer Aided Design
NURBS	–	Non-uniform rational basis spline
VRUT	–	Virtual reality universal toolkit
IGES	–	Initial Graphics Exchange Specification
GUI	–	Graphical user interface
BVH	–	Bounding volume hierarchy

Obsah

1 Úvod

Většina strojních a průmyslových součástek je dnes navržena pomocí CAD modelovacích nástrojů. Základním geometrickým prvkem těchto systémů jsou křivkami ořezané NURBS plochy, které poskytují pohodlný způsob, jak popsat plochy téměř všech tvarů s poměrně malými paměťovými požadavky. Navíc díky ořezávání můžeme jednoduše dostat i velmi komplikované tvary. Kombinací velkého množství těchto ploch můžeme modelovat velmi složité objekty jako jsou automobily či letadla, kde vizualizace těchto dat spoří nemalé náklady na vývoj samotného produktu.

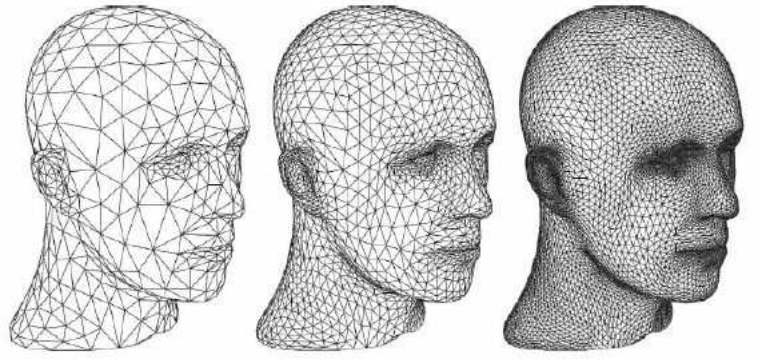
Přesto, že dnes již existují metody na vykreslení NURBS ploch pomocí sledování paprsků, jsou tyto metody přes svou vysokou přesnost a kvalitu pro dnešní hardware stále výpočetně náročné a jejich nasazení je dnes většinou možné jen při finální prezentaci produktu například pro marketingové účely. Pro ostatní části samotného vývoje, kde se hledí především na rychlost vykreslení dané části, se využívá „trojúhelníkový rendering“, jenž je tvořen trojúhelníkovou sítí, která se přibližuje původnímu povrchu. Vedle rychlosti je hlavní předností tohoto přístupu v dnešní době již pokročilá hardwarová optimalizace.

V poslední době je tesselace poměrně často zmiňována s příchodem nového DirectX 11 a jeho podporou v grafických kartách, jenž jí dodávají nový rozměr především v herní grafice. Ovšem zde se nejedná o převod plochy na trojúhelníkovou síť, ale pouze o zjemnění sítě původní. Jinak řečeno, vezme každý trojúhelník a rozdělí ho na několik menších, čímž celému modelu dodá detail hrbolatosti.

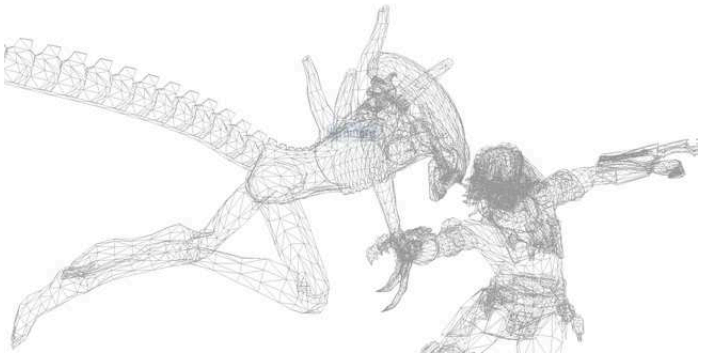
Úkolem mé diplomové práce bylo naleznout a implementovat optimální metody pro realizaci tesselace v programu VRUT. Cílem bylo nalézt takové řešení, které by ve výsledku odpovídalo výsledkům (výsledný vzhled trojúhelníkové sítě, počet trojúhelníků) a rychlostí komerčních nástrojů (RTT DeltaGen, VRED). Přestože tyto nástroje obsahuji dostatečně výkoné a přesné tesselátory, liší se jednotlivé programy v dalších různých vlastnostech, kdy jedna nebo druhá je potřebná k využívání, což vede k nutnosti zakoupení licencí na jednotlivé typy programů a tedy i k větším nákladům. Cílem VRUTU je tyto vlastnosti integrovat do jednoho programu a ušetřit čas a celkové náklady při vývoji. Dalším důvodem proč tato práce vznikla je ten, že i přes veškerou snahu dnešní nástroje nedokáží pokrýt všechny zvláštní případy, které mohou při tesselaci nastat. To ve výsledku vede k reklamaci u dodavatele a k případnému čekání na vydání opravy, jenž může vést k tragickým zpožděním při vývoji, neboť vydání opravy je v plné režii zhotovitele produktu, který může mít nastaveny priority jinak než požaduje zákazník. Proto vzniká alternativa ve formě VRUTu, kdy jakýkoliv problém může byt vykonzultován a opraven v podstatně kratší době než je u komerčních produktů. Poslední slabinou komerčních programů je šicí algoritmus, kde výsledné sešití neodpovídalo vždy daným představám. Sekundárním úkolem bylo vytvořit parserovací modul, který by dokázal načíst a uložit do jádra VRUTU formát IGES. Hlavním cílem zde bylo realizovat zpracování tak, aby se rychlostně vyrovnalo či dokonce předběhlo komerční aplikace.

Proces tesselace lze rozdělit na tři části, skládající se z aproximace křivky a plochy dle zadaného tolerančního kritéria, která nám přinese množinu bodů definující daný objekt, který následně přetriangulujeme tzn. vytvořím tzv. trojúhelníkovou síť, reprezentující

daný objekt. Jak je rovněž z poslední věty vidět, tak často zaměňované pojmy triangulace a tesselace nejsou dva tytéž výrazy, ale dva odlišné postupy. Tyto dva kroky ovšem vedou k vytváření tzv. „děr“, které jsou odstraněny posledním krokem jemuž se většinou říká „sešití“.



Obrázek 1: Zjemňování trojúhelníkové sítě pomocí hardwarové tesselace [14]

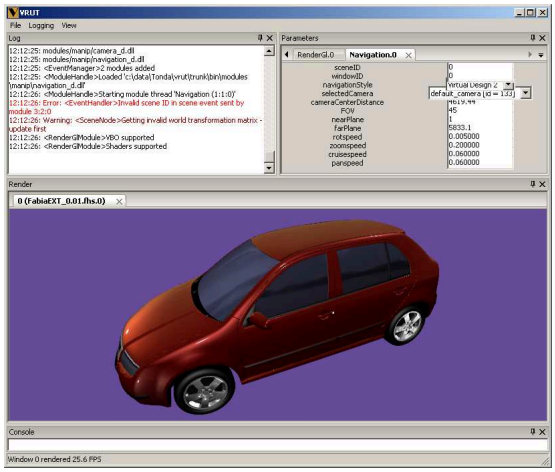


Obrázek 2: Ukázka tesselace ze hry Alien vs. predator 2 [14]

1.1 Vrut

Jedná se o aplikaci určenou pro vizualizaci a editaci 3D, která si klade za cíl:

- využít nových technologií
- poskytnuti zvláštní funkcionality
- podpora systémů a formátů Škoda auto
- vysoká rychlost
- použitelnost pro výuku a experimenty



Obrázek 3: Pohled na GUI VRUTu

Projekt s názvem VRUT vznikl v rámci spolupráce katedry počítačové grafiky a interakce ČVUT FEL s firmou Škoda Auto. Jeho podstatou je zobrazení grafických a podpora modulů. Moduly umožňují rozšíření funkcí hlavní aplikace při relativní nezávislosti na ní. Název VRUT je zkratkou anglického Virtual Reality Universal Toolkit a měl by zhruba evokovat univerzální a flexibilní nástroj pro práci s grafickými daty. VRUT může fungovat jako určitý spojovací článek mezi několika subjekty. Jednotlivé subjekty mohou zadávat různé úlohy týkající se zobrazení nebo jiných operací nad grafickými daty a řešitelé těchto úloh vytvoří kompatibilní modul (plugin).

VRUT nelze jednoznačně zařadit do určité kategorie software. Přesto existují komerční aplikace jenž sloužily jako inspirace, na jejichž základě byl VRUT konstruován. Jedná se o aplikace VRCOM Virtual Design 2, IC:IDO VDP, RTT DeltaGen, VRED a další.

Jak jsem již jednou zmínil, jedná se o modulární aplikaci, využívající pro svůj chod moduly. Aplikace je rozdělená na dvě části a to na jádro a balíček modulů. Samotné jádro je z uživatelského hlediska nepoužitelné, neboť primární funkcí jádra je správa modulů a správa grafických dat, dále pak jádro poskytuje pomocné prvky pro správný běh a

ovládání aplikace. Další podstatnou části jádra je také správa událostí tvořící centrální prvek komunikačního kanálu mezi jednotlivými částmi aplikace a moduly.

Hlavním smyslem, proč byla zvolena modulární aplikace, je možnost rozšiřování o nové funkce. Moduly jsou aktivovány a spravovány příslušným správcem modulů. V jeden moment můžou být aktivní všechny dostupné moduly a navíc libovolné množství instancí od jednotlivých modulů. Každému modulu respektive každé instanci je přiřazeno vlastní vlákno, ve kterém probíhá veškerá činnost modulu. Kompletní popis funkcionality a architektury je možno dohledat v [2].

2 IGES a jeho zpracování

Jedná se o grafický formát vytvořený v roce 1979 firmami Boeing, General Elektrix, Xerox za podpory National Bureau of Standards (dnes NIST) a amerického ministerstva obrany. Účelem vzniku byl vznik stdandardizovaného formátu pro elektronickou podobu a přenos konstrukčních výkresů. Během svého vývoje vzniklo několik verzí. Aktuální verze tohoto formátu má číslo 5.3 a je plně akreditována jako ANSI standard. Celý formát je kompletně a přehledně popsán v této specifikaci [1], která je ovšem dosti rozsáhlá a proto pro přiblížení vyberu jen nejzákladnější informace.

2.1 Struktura IGES souboru

Informace mají v tomto souboru buďto textový nebo binární formát. Mým úkolem bylo pouze zpracování textového formátu a tudíž zde budu popisovat pouze jeho. Struktura IGES souboru se skládá z následujících částí

- start (S)
- global (G)
- direction entry (D)
- parameter data (P)
- terminate (T)

kde jednotlivé sekce začínají v souboru vždy na novém řádku. Písmeno definované vedle názvu sekce slouží jako identifikátor a je umístěn vždy jako předposlední hodnota daného řádku. Předposlední hodnotou je číslo řádku, jenž je definováno v rámci dané sekce. Sekce *start* slouží jen a pouze jako komentář autora a nenese tak sebou žádnou důležitou informaci. Tak samo není důležitá ani sekce terminate, která dle dokumentace slouží jen jako jakási sumarizace celého souboru.

```
1G      4D      36P     32                                      T       1
```

Výpis 1: Uázka zápisu sekce *terminate*

Global sekce obsahuje informace o daném souboru (název a ID souboru, datum poslední změny, typ a název měrných jednotek, typ oddělovače, jméno autora a mnoho dalších). Jednotlivé hodnoty jsou buďto datového typu Integer nebo String. Velikost jednotlivých prvků není znakově omezena a jejich rozpoznávání závisí na typu oddělovače, což je ve většině případů čárka.

Directory entry sekce obsahuje právě jeden záznam pro každou jednotlivou entitu. Její délka je fixní, kde každý prvek může obsahovat maximálně 8 znaků. Každý záznam je pak rozložen do dvou řádků (po 80 znacích na řádek). Co se týče funkcionality této sekce, tak ji lze chápat jako ukazatel, jenž drží informace o umístění entity v souboru,

kolik řádků daná entita zabírá, o jaký typ entity se jedná atd.

```
128     10      0       0       0       0       0       001010001D  11
128      0      0       3       0                               0D  12
```

Výpis 2: Uázka zápisu sekce *directory entry*

Poslední *parametric* sekce obsahuje parametrická data, reprezentující danou entitu. Sama o sobě nemá stanovenou pevnou délku jednotlivých prvků, neboť struktura se pro každou entitu liší v závislosti na typu geometrického objektu. I přes tuto zdánlivou volnost, jsou zde vymezeny některé sloupce, které slouží ke snadnější orientaci v souboru. Konkrétně sloupec 66 – 72 odkazuje na číslo řádku přidružené Directory entry sekce. Ve sloupci 74 – 80 je definováno čislo řádku v dané sekci. Jak jsem již zmínil jedná se zde především o definic jednotlivých entit, kde každý popis začíná číslem jenž definuje typ entity. Konec definice je vždy definován středníkem.

```
126,1,1,0,0,1,0,−35.0,−35.0,35.0,35.0,1.0,1.0,−100.0,40.0,0.0,      1P      1
−100.0,−30.0,0.0,−35.0,35.0,0.0,0.0,0.0,0.0,0;                      1P      2
```

Výpis 3: Uázka zápisu sekce *parametric*

Nyní tedy mohu přistoupit popisu vybraných entit.

2.2 Popis vybraných entit

Entitou se v tomto formátu může nazývat samotný geometrický prvek (křivka, plocha, úsečka, ...) nebo v případě složitějších konstrukčních objektů se jedná i o prvky, které udržují vazby na jednotlivé geometrické útvary. V případě plochy tvořené ořezovými křivkami, drží entita vazby jednak na samotnou plochu, ale také na potřebné křivky a další vlastnosti jako jsou například transformace.

Množství entit, které IGES popisuje a podporuje je tak veliké, že jeho celkové pokrytí nebylo časově realizovatelné. Proto můj parser podporuje zpracování, jen těch entit, které byly potřebné k další realizaci mé práce s tím, že další potřebné entity můžou být dle potřeb přidány. Pro každou entitu je vytvořena vlastní struktura, jejíž parametry jsou závislé na tom jak je definována v manuálu. Tyto struktury jsou součástí datového typu GeometryTriangles, jenž slouží jako základ pro jakoukoliv geometrii, tvořenou trojúhelníkovou sítí, uloženou v jádře VRUTu. NURBS křivku nalezneme jako entitu číslo 126 a plochu jako 128. Jejich samotná definice není nijak zajímavá a proto ji zde vynechám a zaměřím se rovnou na entity definující geometrické objekty z těchto dvou prvků.

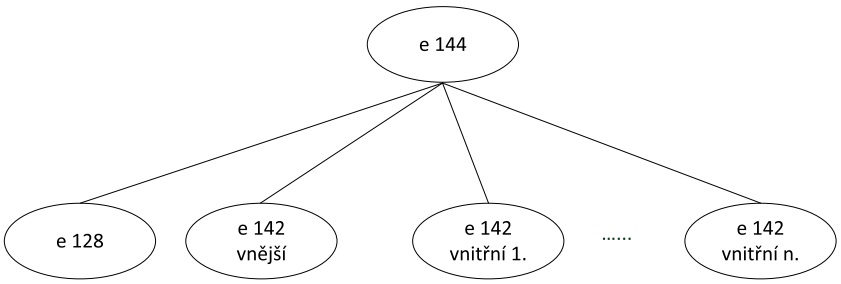
2.2.1 Entita definující ořezanou plochu

Jak již bylo a ještě mnohokrát bude zmíněno, složitější tvary nelze jednoduše popsat pomocí samotné plochy, ale je k tomu potřeba využít ořezových křivek. Dle manuálu má tato entita přiděleno referenční číslo 144 a v mém programu je definovana takto.

```
struct TrimmedSurface
{
    int pts, n1, pto;
    std::vector<int> pti;
    std::vector<int> deNa, deNp;
    int dePointer;
};
```

Výpis 4: Datová struktura definující entitu č. 144

kde proměnné `pts, pto, pti` slouží jako indexový ukazatel do sekce *Directory entry*. První z nich je ukazatel na plochu (č. 128), druhá entitu definující vnější ořezovou křivku. Třetí zmiňovaná proměnná pak definuje entity pro vnitřní ořezové křivky (pro vnější i vnitřní křivky je to entita číslo 142), kde parametr `n1` definuje jejich celkový počet. Dále je si možno z definice povšimnout proměnných začínajících písmeny `de`. Tyto proměnné jsou definovány jako další vlastnosti prvku s tím, že se jedná rovněž o ukazatele do sekce *Directory entry*. Jelikož jsem se během realizace a testování svého programu nesetkal s entitou, která by tyto vlastnosti využívala, proto se nebudu již dále o nich více zmiňovat a pro jejich hlubší pochopení doporučuji nastudovat [1].



Obrázek 4: Grafické znázornění entity 144

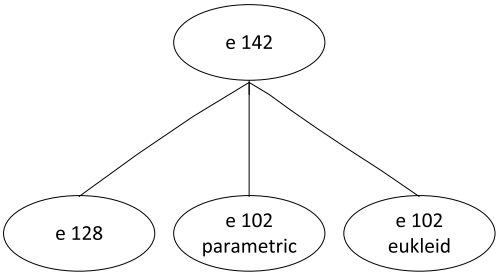
2.2.2 Entita definující ořezové křivky

Primární vlastností této entity, s definujícím číslem 142, je určení, které ploše daná křivka náleží. Struktura této entity je definována následovně

```
struct CurveOnParametricSurface
{
    int crtn, sptr, bptr, cptr, pref;
    std::vector<int> deNa, deNp;
    int dePointer;
};
```

Výpis 5: Datová struktura definující entitu č. 142

kde pro nás důležitými prvky jsou proměnné `bptr` a `cptr`. `bptr` je pointer na entitu číslo 102 definující křivku v parametrickém prostoru. Na druhou stranu proměnná `cptr` obsahuje odkaz na tutéž entitu s tím rozdílem, že se jedná o křivku, definovanou v eukleidovském prostoru. Posledním zajímavým parametrem je proměnná `sptr`, definující plochu, na které daná křivka leží.



Obrázek 5: Grafické znázornění entity 142

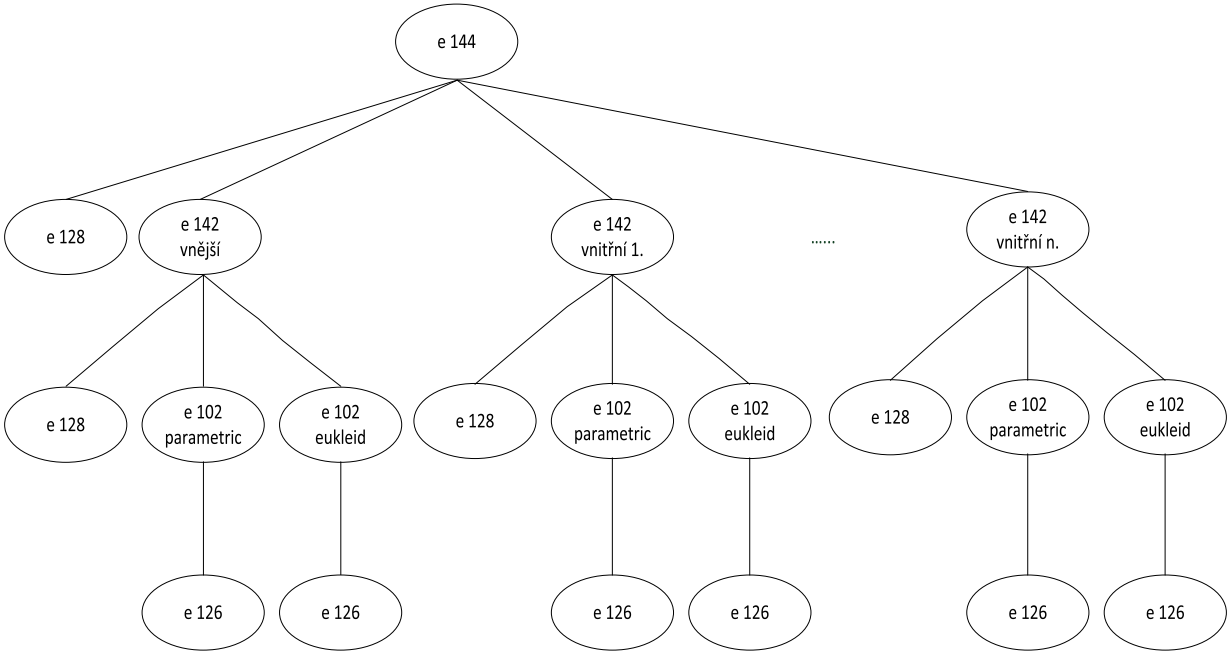
2.2.3 Entita Popisující kompositní křivku

Jak již lze z názvu této podkapitoly vyčíst jedná se o jednoduchou entitu definující komplexní křivku, která se může skládat z různých prvků jako jsou úsečky, kruhové oblouky, spline křivky a další. Číselně je tato entita označena jako 102. Struktura objektu definující tuto entitu v jádře VRUTu je popsána nálsedovně

```
struct CompositeCurve
{
    std::vector<int> de;
    int dePointer;
};
```

Výpis 6: Datová struktura definující entitu č. 102

kde jediným důležitým prvkem je pole ukazatelů `de`, které odkazuje na liniové objekty do *directory entry* sekce. Na obrázku 6 je graficky zobrazena stromová struktura, která definuje obdelnikovou rovinu včetně svých ořezových křivek.



Obrázek 6: Výsledný strom popisující ořezanou plochu

3 NURBS křivky a plochy

NURBS křivky a plochy jsou v současnosti zřejmě nejpoužívanější reprezentací obecných křivek a ploch, používaných v řadě aplikací, zahrnujících oblasti desingu a navrhování. V základu NURBS reprezentuje zobecnění B-splinů. Neuniformní B-spline je tedy B-spline jehož uzlový vektor obsahuje alespoň jednu dvojici t_i, t_{i+1}, jejíž vzdálenost se liší od ostatních. Racionalita znamená, že každému bodu je přiřazena jeho váha. Ta určuje jakou silou bude bod působit na tvar křivky resp. plochy. Pokud mají všechny body váhu konstantní, pak se jedná o B-spline křivku resp. plochu. Přidáním vah k bodům, které nejsou pro všechny body konstantní, se dostáváme do projektivního prostoru. Na rozdíl od jiných dosud známých křivek zde můžeme také přesně zadávat kuželosečkové oblouky.

3.1 B-spline bázové funkce

Význam bázových funkcí spočívá v reprezentaci interpolačního nebo aproximačního schématu vyjadřujícího vztah mezi průběhem křivky a řídícím polygonem. Vlastnosti tohoto schématu jsou dány volbou bázových funkcí. B-spline bázová funkce s uzlovým vektorem $\mathbf{u} = (u_0, \ldots, u_{p+n+1})$ je definována následovně

$$N_i^0(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & \text{ostatní} \end{cases} \tag{1}$$

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1}^{p-1}(u)\,, \tag{2}$$

kde p značí stupeň funkce a pro $0 \leq i \leq n-p-1, 1 \leq p \leq n-1$. Počet uzlů t_i je roven $n+1$.

Některé vlastnosti bázových funkcí:

- pro výpočet *i*-té bázové funkce potřebujeme znát uzlový vektor $\mathbf{u}$ a stupeň bázové funkce p
- funkce $N_i^p(u)$ je nezáporná pro libovolné i, p a u
- součet všech bázových funkcí stupně p na intervalu je roven 1

$$\sum_{j=i-p}^{i} N_j^p(u) = 1 \qquad u \in \langle u_i, u_{i+1})$$

- Označíme-li počet uzlů jako $m + 1$ a počet bázových funkcí stupně p pomocí $n + 1$, pak platí $m = n + p + 1$.

Pro další informace o bázových funkcích doporučuji nastudovat [4]. Názorný a srozumitelný příklad výpočtu bázových funkcí lze nalézt v [3].

3.2 NURBS křivky

NURBS křivka stupně p je na intervalu $u \in \langle a, b)$ definována následovně,

$$\mathbf{C}(t) = \frac{\sum_{i=0}^{n} \mathbf{P}_i w_i N_i^p(t)}{\sum_{i=0}^{n} w_i N_i^p(t)}\,, \tag{3}$$

kde $\mathbf{P}_i$ je $n+1$ řídicích bodů, $w_i > 0$ jsou váhy přidružené k jednotlivým bodům, $N_i^p(t)$jsou bázové funkce stupně p definované nad parametrizací neperiodickým a neuniformním uzlovým vektorem $\mathbf{t} = (t_0, \ldots, t_{p+n+1})$. Existuje také zápis využívající racionálních bázových funkcí,

$$R_i^p(t) = \frac{w_i N_i^p(t)}{\sum_{k=0}^{n} w_k N_k^p(t)}\,, \tag{4}$$

kdy můžeme naši rovnici přepsat na následující tvar

$$\mathbf{C}(t) = \sum_{i=0}^{n} \mathbf{P}_i R_i^p(t)\,. \tag{5}$$

Vlastní body NURBS křivky v projektivní rovině jsou dány třemi souřadnicemi, které určují souřadnice bodu euklidovské roviny a váhu bodu. Vlastní body prostorové křivky jsou reprezentovány čtyřmi homogenními souřadnicemi (x_i', y_i', z_i', w_i), které odpovídají euklidovským bodům, kde w_i značí váhu daného bodu. V mém případě jsem ovšem počítal s křivkami v prostoru parametrickém, kde místo souřadnic (x_i, y_i, z_i) dostanu jen koordináty u, v , které mi umožňují počítat s křivkami ve dvourozměrném prostoru. Potřebné souřadnice pák získám dosazením u, v do rovnice pro NURBS plochu.

Další vlastnosti NURBS křivek

- jsou invariantní vůči transformacím, rovnoběžnému a středovému promítání
- umožňují přesné vyjádření kuželoseček

Jistou zvláštností oproti Bézierovým křivkám je to, že parametr zde leží na otevřeném intervalu. Tato vlastnost způsobuje problémy především při výpočtu krajního bodu křivky s parametrem, který odpovídá koncovému intervalu, ale jelikož je otevřený nemůže být vypočten dle zmíněných vzorců. Řešení je ovšem velice jednoduché, neboť místo vypočítané hodnoty vrátíme hodnotu posledního řídícího bodu, který s tímto parametrem koresponduje. Toto pravidlo je ovšem použitelné za předpokladu, že uzlový vektor končí (a začíná) $p+1$ násobným uzlem. Než jsem se o této správné možnosti dozvěděl, řešil jsem tento problém odečtem velmi malého čísla např. 0.0001, což bylo pouhým okem nepostřehnutelné, ale ve výsledných geometriích se v některých případech vyskytovaly nezanedbatelné nepřesnosti. Navíc zde existuje riziko porušení pravidla, kdy uzlový vektor bude neklesající posloupnost.

3.3 NURBS plochy

NURBS plocha stupně p ve směru u a stupně q ve směru v, je dána sítí $(n+1)(m+1)$ kontrolních bodů $\mathbf{P}_{i,j}$ a váhami $w_{i,j}$, které jsou přidruženy k jednotlivým bodům,

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\mathbf{P}_{i,j}w_{i,j}N_i^p(u)N_j^q(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m}w_{i,j}N_i^p(u)N_j^q(v)}\,, \tag{6}$$

kde N_i^p, N_j^p jsou bázové funkce a $\mathbf{u}, \mathbf{v}$ jsou uzlové vektory. Stejně jako u křivky i zde můžeme využít zápis pomocí racionální bázové funkce,

$$R_{i,j}^{p,q}(u,v) = \frac{w_{i,j}N_i^p(u)N_j^q(v)}{\sum_{k=0}^{n}\sum_{l=0}^{m}w_{k,l}N_k^p(u)N_l^q(v)}\,, \tag{7}$$

kde dosazením následně získáme vztah

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m}\mathbf{P}_{i,j}R_{i,j}^{p,q}(u,v)\,. \tag{8}$$

NURBS plochy mají vlastnosti přenesené z NURBS křivek, jenž jsou rozšířené do prostoru. Opět uvedu výběr několika základních vlastností:

- B – spline funkce jsou nezáporné
- body NURBS plochy leží v konvexním obalu sítě řídících bodů
- změna polohy libovolného řídicího bodu nebo váhy změní tvar plochy pouze na příslušném intervalu
- NURBS plocha je invariantní vůči projektivním transformacím

Stejně jako u B-spline funkcí se zde jedná jen o výběr nejzákladnějších informací. Pro komplexnější přehled doporučuji nastudování [3, 4].

Podobně jako u křivek i zde nastává problém v okamžiku, kdy na vstupu obdržím hodnotu parametru, který odpovídá otevřenému intervalu. Zde je problém navíc umocněn tím, že se nejedná pouze o rohové body, ale i o dvě hrany, které se na tomto otevřeném intervalu objevují. Řešení rohových bodů je obdobné jako u křivek, kdy stačí dosadit hodnoty odpovídajících rohových řídících bodů. Co se týče hran musím zde využít znalosti o tom, že NURBS plocha může být reprezentována jako tenzorový součin křivek, a tedy mohu inkriminované hrany počítat jako křivky, kdy řídící body dané křivky odpovídají řídícím bodům, jenž leží ve směru počítané hrany. Parametr t a uzlový vektor odpovídá zvolenému směru.

4 Aproximace křivek a ploch

Aproximace je proces, při kterém prokládáme množinu bodů aproximačním polygonem. Důvod proč aproximujeme křivku či plochu je ten, že chceme redukovat počet bodů potřebných k co nejpřesnějšímu vykreslení a tudíž i k zvýšení vykreslovací a výpočetní rychlosti. Tímto krokem v podstatě začíná celá tesselace, kdy na základě zpracované geometrie datového typu `Geometry` obdržíme informace definující plochu a k ní odpovídající vnitřní a vnější ořezové křivky. Vnější ořezové křivky jsou v `Geometry` uloženy v poli, určeném pro křivky, vždy na indexu 0. Vnější i vnitřní křivky se většinou skládají z více křivek. Aproximace se však vypočítává pro každou křivku zvlášť, kdy na výstupu očekáváme množinu bodů, jimiž prochází naaproximovaný polygon. U ploch se postupuje obdobným způsobem jako u křivek s tím, že na vstup pošleme vždy jen jednu plochu. Výstupem je zde množina rovinných bodů. Námi získané výstupy poté přetriangulujeme.

4.1 Bézierova křivka

Bézierova křivka n-tého stupně pro $n+1$ kontrolních bodů, které tvoří tzv. řídící polygon, je pro $t \in \langle 0, 1 \rangle$ definována jako

$$\mathbf{C}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{P}_i \,, \tag{9}$$

kde je $B_i^n(t)$ i-tý Bernsteinův polynom n-tého stupně

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-1} \,. \tag{10}$$

Bernsteinovy polynomy tvoří bázi vektorového prostoru polynomu a splňují rekurentní vzorec

$$\begin{aligned} B_i^n(t) &= \begin{cases} 0 & \forall (i<0) \vee (i>n) \\ (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) & \text{ostatní} \end{cases} \\ B_0^0(t) &= 1 \,. \end{aligned} \tag{11}$$

Vlastnosti:

- křivka se nachází uvnitř konvexní obálky svého řídícího polygonu, který je definován svými řídicími body
- křivka prochází koncovými body $\mathbf{P}_1$ a $\mathbf{P}_n$
- leží-li všechny kontrolní body na jedné přímce, pak se Bézierová křivka stává úsečkou
- je-li $n = 1$, pak křivka je tvořena dvěma kontrolními body a rovněž se jedná o úsečku

4.2 Bézierova plocha

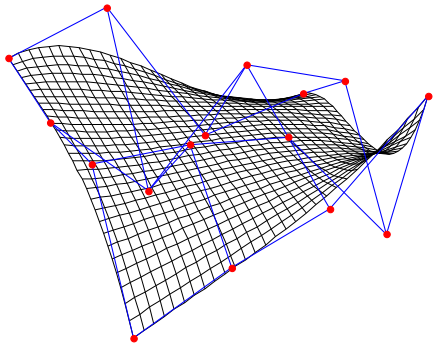
Rozšířením Bézierovy křivky získáme Bézierovu plochu, která patří mezi parametrické plochy určené sítí bodů. Výslednou plochu o velikosti $m \times n$ vypočteme jako

$$\mathbf{S}(u,v) = \sum_{j=0}^{m} \sum_{i=0}^{n} \mathbf{P}_{i,j} B_i^n(u) B_j^m(v), \quad \text{kde } u, v \in \langle 0, 1 \rangle \tag{12}$$

$$B_i^k(t) = \binom{k}{i} t^i (1-t)^{k-i}, \quad i = 0, 1, \ldots, k\,. \tag{13}$$

Vlastnosti:

- Bézierova plocha prochází svými čtyřmi rohovými body sítě
- okrajové řídící body jednotlivých stran zadané sítě bodů tvoří také zadání pro Bézierovy křivky. Plocha je tedy ohraničena (nikoli zadaná) čtyřmi Bézierovými křivkami
- všechny body plochy leží v konvexním obalu zadaného pomocí řídící sítě



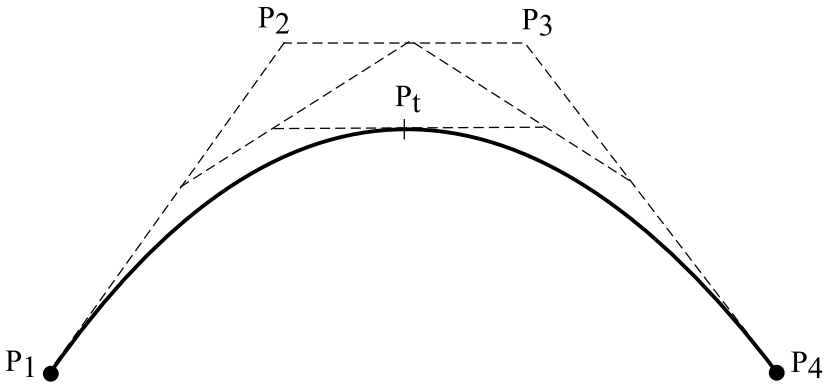
Obrázek 7: Ukázka Bézierovy plochy a jeho řídících bodů [19]

4.2.1 De Casteljau algoritmus

Jedná se o rekurzivní metodu, která slouží k výpočtu bodu na Bézierove křivce. V podstatě se u tohoto algoritmu nejedná o nic jiného, než o postupné dělení úsečky řídícího polygonu v zadaném poměru. Počet nově vzniklých bodů se vždy snižuje o jeden až do doby, kdy nám jeden jediný zůstane. V tomto případě se jedná o nalezení hledaného bodu. Matematicky lze tento algoritmus vyjádřit takto

$$\mathbf{C}_j^0(t) = \mathbf{P}_i \tag{14}$$

$$\mathbf{C}_j^i(t) = (1-t)\,\mathbf{C}_i^{j-1}(t) + t\,\mathbf{C}_{t+1}^{j-1}(t) \begin{cases} \text{j = 1, \ldots, n} \\ \text{i = 0, \ldots, n - j} \end{cases}, \tag{15}$$



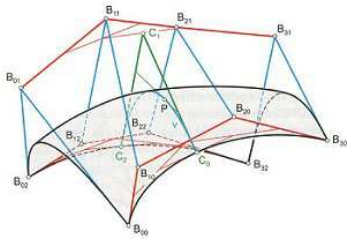
Obrázek 8: Geometrický popis algoritmu de Castljau

kde n je počet řídících bodů dané křivky. Naprogramování tohoto algoritmu je velice jednoduché a lze ho popsat následujícím pseudo kódem:

```
for m = 1 to n do
{
   for j = 1 to n – m do
   {
      P_j^(m) = t P_(j+1)^(m-1) + (1 − t) P_j^(m-1)
   }
}
```

Výpis 7: Algoritmus de Castljau

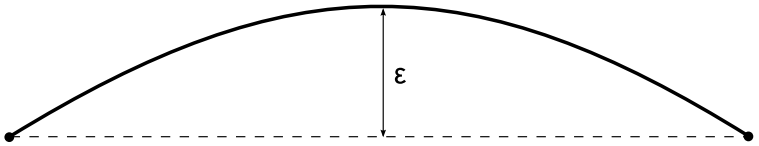
Obdobným způsobem lze využít tento algoritmus i pro plochu. Dobře popsaný postup lze nalézt v [16]



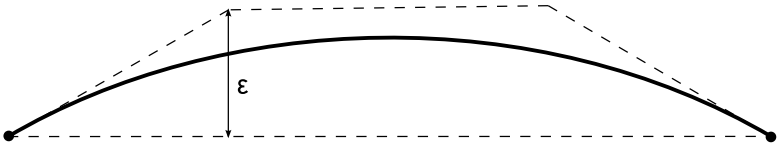
Obrázek 9: Algoritmus de Castljau aplikovaný na Bézierově ploše

4.3 Aproximace křivek

Abychom dostali plochu, která má složitější tvar a strukturu, tak místo složitého modelování této plochy ji můžeme ořezat pomocí křivek. V zájmu zachování vysoké rychlosti se nevykresluje přesná křivka, ale pouze otevřený polygon, který se snaží k této křivce co nejvíce přiblížit. K tomu, abychom stanovili, jak velice se bude polygon od křivky odlišovat, je stanoveno toleranční kritérium ϵ. Toto kritérium lze rovněž definovat jako největší vzdálenost bodu křivky daného úseku od hrany polygonu viz. obrázek 10. Pokud se k aproximaci využívá převodu na Beziérovy křivky, pak můžeme toleranci definovat jako vzdálenost kontrolních bodů této křivky od úsečky, která je definována počátečním a koncovým bodem této křivky nebo jinak řečeno jedná se o konvexní obal, který ji obklopuje s danou tolerancí ϵ viz. 11. Důvod proč se používá tato méně přesná metoda je ten, že není příliš jednoduché naleznout místo s největší vzdálenosti od dané úsečky viz. nativní metoda 4.3.1. Toleranční kritérium se v grafických programech většinou stanovuje na základní hodnotu 0.2 s tím, že nižší hodnota znamená vyšší přesnost.



Obrázek 10: Definice tolerance



Obrázek 11: Kontrola tolerance pomocí řídících bodů Beziérovy křivky

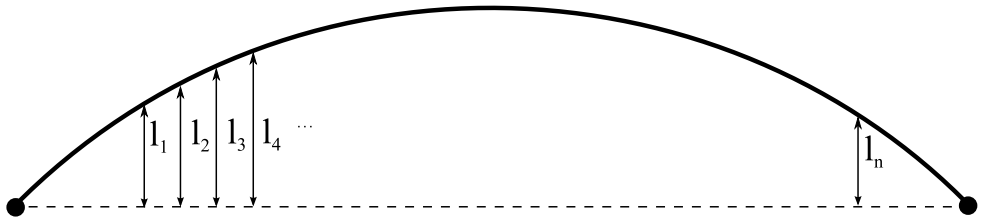
4.3.1 Nativní metoda

Tato metoda byla zpočátku zvolena díky své jednoduché implementaci a téměř nulové teoretické připravě jako testovací pro ověření funkcionality zbývajících algoritmů. Tento algoritmus je implementován v metodě `aproximateCurve()`, kde na vstupu obdržím počáteční a koncový parametr křivky, jenž odpovídá buďto celé křivce nebo jen jeho části a to z důvodu rekurentního využití této metody. Na výstupu obdržím aproximovaný polygon přibližující se originální křivce.

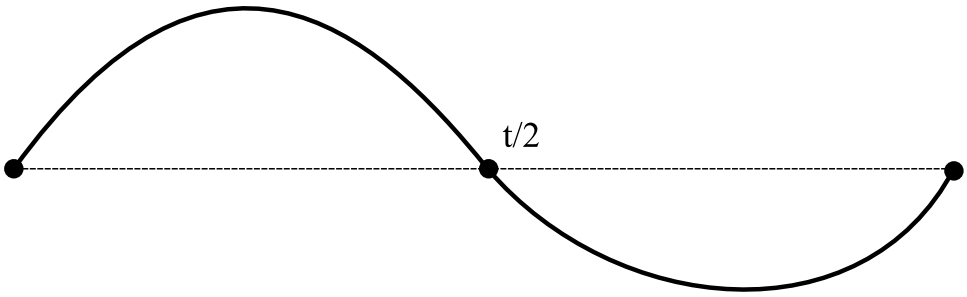
Funkce algoritmu je následující. Pro parametr křivky $t/2$ vypočítáme jeho souřadnicí. Nutno podotknou, že dosazením parametru do rovnice křivky dostaneme pouze souřadnice odpovídající dvourozměrné parametrické rovině, kterou ještě musíme převést na hodnoty 3D prostou, které získáme dosazením do rovnice plochy. Na základě získané souřadnice vypočtu vzdálenost tohoto bodu od úsečky, která odpovídá

počátečnímu a koncovému bodu křivky. Pokud vypočtená vzdálenost je větší než povoluje tolerance, pak rekurzivně volám metodu `aproximateCurve()`, kterou použiju dvakrát pro rozdělený vstupní interval např. $\langle t/2, t\rangle$.

Pokud vypočtená vzdálenost odpovídá toleranci, pak musíme ještě provést následující test. Vstupní interval je rozdělen na veliké množství malých částí (kroků). V našem algoritmu se jednalo o tisícovku těchto kroků, kdy čím větší je jejich počet, tím menší je riziko chyby, která spočívá v nenalezení lokálního minima mezi dvěma kroky. Pro každou část je vypočtena opět vzdálenost od úsečky a pokud odpovídá toleranci přejdeme na výpočet dalšího kroku, v opačném případě rozdělíme vstupní interval na dvě poloviny a opět rekurzivně voláme metodu `aproximateCurve()`.



Obrázek 12: Ukázka výpočtu tolerance



Obrázek 13: Ukázka zvláštního případu, kvůli němuž je potřeba dělit interval na tak malé kroky

4.3.2 Aproximace převodem na Beziérovy křivky

Důvod proč byla nahrazena původní nativní metoda je hlavně vyšší rychlost a absence možnosti výskytu chyby zmiňované v předchozí kapitole. Postup, při kterém křivku převedeme na otevřený polygon dle zadané tolerance, lze rozdělit na dvě části:

- převod NURBS křivky na Beziérovy křivky
- dělení jednotlivých křivek na jemnější segmenty, splňující toleranci

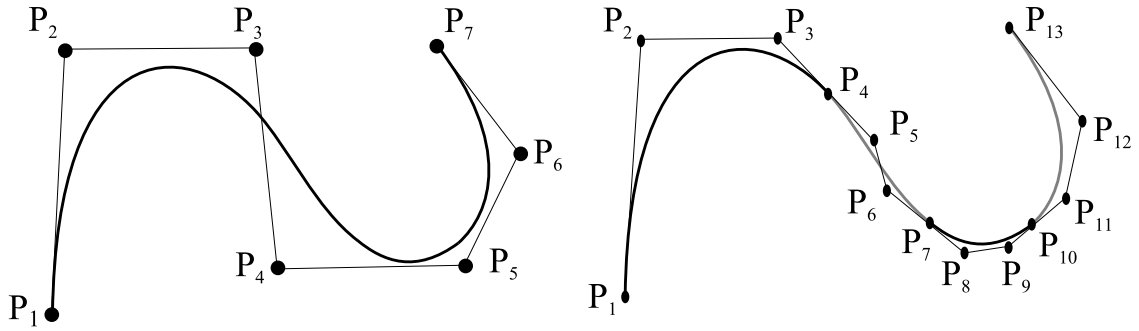
4.3.2.1 Převod NURBS křivky na Beziérovy křivky

Princip převodu je odvozen od De Boorova algoritmu [17], který vychází z algoritmu de Casteljau. Detailní kód k algoritmu naleznete v [5] s tím, že hned na začátku tohoto kódu se vyskytuje chyba, která způsobuje při implementaci pád programu. K odstranění této chyby stačí místo $m = n + p + 1$ pouze dosadit $m = n + p$.

V rámci urychlení tohoto algoritmu využijeme jednu z vlastností Bézierovy křivky a to tu, že pokud na vstupu obdržíme pouze dva řídící body, tak dále nepokračujeme, neboť se jedná o úsečku, definovanou jejími dvěma řídícím body. Tento případ se u složitějších objektů objevoval poměrně často, proto je využití této vlastnosti vítáno jako zrychlující krok.

Výsledná Béziérová křivka má tyto vlastnosti

- stupeň m je shodný s původní NURBS křivkou
- počet řídících bodů každé Bézierové křivky odpovídá hodnotě $m + 1$

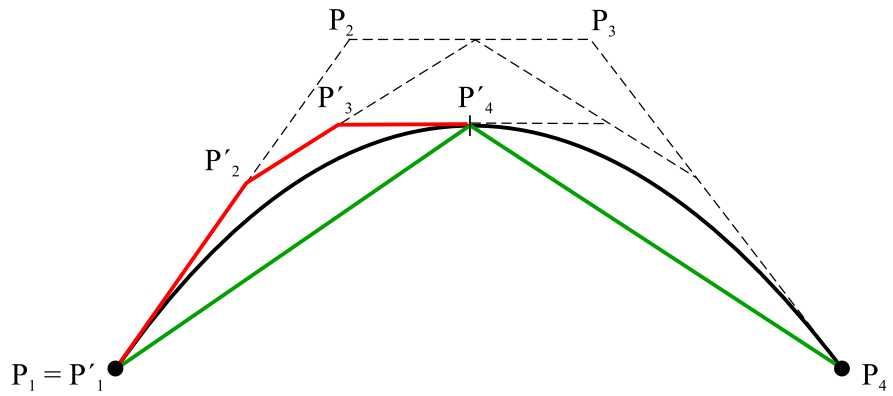


Obrázek 14: Převod NURBS křivky třetího stupně (vlevo) na sadu navazujících Beziérových křivek [13]

4.3.2.2 Algoritmus dělení Beziérových křivek dle zadané tolerance K dělení Bézierových křivek, které jsme dostali v předchozí kapitole, využijeme známý algoritmus de Castljau. Každou křivku budeme tímto algoritmem rekurzivně dělit na polovinu dokud vzdálenost řídících bodů od úsečky, která je definována počátečním a koncovým řídícím bodem, není menší než zadaná tolerance ϵ. Jinak řečeno jedná se zde o šířku konvexního obalu, který danou křivku obklopuje. Každým novým dělením dochází k vytváření nových řídících bodů, které se ke křivce postupně přibližují a čim dál více ji opisují viz. 15. Popis použitého algoritmu je následující:

Na vstupu: řídící body Beziérovy křivky $\mathbf{P}i, \ldots, n$
Na výstupu: seznam nově vytvořených Beziérových křivek splňující toleranci

1. spočítej maximální vzdálenost $d = \max(\mathbf{P}_i, l(\mathbf{P}_0, \mathbf{P}_n))$
2. jestliže $d <$ tolerance ϵ ulož do seznamu a ukonči algoritmus v opačném případě pokračuj na 3
3. rozděl křivku algoritmem de Castljau s polovičním vstupním parametrem na dvě křivky a rekurzivně pokračuj na bod 1



Obrázek 15: Dělení Beziérovy křivky a vznik nových kontrolních bodů (červeně). Zelenou barvou vznikající aproximační polygon

Tento algoritmus se provede pro všechny Beziérovy křivky, které jsme získali převodem z NURBS křivky. Po dělení a kontrole všech křivek dostaneme seznam na jehož základě obdržíme výsledný polygon. Z definice algoritmu de Castljau plyne, že na původní křivce leží jen počáteční a koncový bod získaných segmentů, tudíž ostatní řídící body ze získaného seznamu můžeme zahodit a dále jen pracovat s těmi odpovídajícími, které tvoří výslednou hranu hledaného polygonu.

4.4 Aproximace ploch

Stejně jako u křivek tak ani u ploch nevykreslujeme skutečnou plochu, ale pouze polygon, který se svou strukturou dané ploše přibližuje. Cílem této kapitoly je převést na vstupu získanou NURBS plochu, definovanou pouze svými vlastnostmi jako jsou řídící body, uzlové vektory váhy atd. viz kapitola věnována NURBS plochám 3.3 , na odpovídající obdélníkovou síť, která nám reprezentuje daný tvar definované plochy. Proto, abychom na výstupu dostali obdélníkovou síť, musela být jednak definována datová struktura `Rectangle` a dále postup dle kterého se bude daná plocha aproximovat. Stejně jako u křivek byla zde z důvodů rychlé implementace nejdříve zvolená nativní metoda, která byla později nahrazena metodou pro převod na Beziérovy pláty. Struktura `Rectangle` je definována takto

```
struct  Rectangle
{
        Vector3 p1;
        Vector3 p2;
        Vector3 p3;
        Vector3 p4;
        std :: vector<Vector2> p1p2Points;
        std :: vector<Vector2> p2p3Points;
        std :: vector<Vector2> p3p4Points;
        std :: vector<Vector2> p4p1Points;
}
```

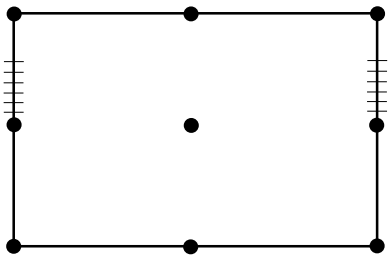
Výpis 8: Definice struktury `Rectangle`

kde proměnné `p1`, ..., `pn` reprezentují rohové body obdélníku. Pole `p1p2Points`, ..., `p1p2Points` se využívají až na konci algoritmu a obsahují všechny body, které danou hranu dělí. Při převodu na tento datový typ vznikají duplicity, které jsou následně odstraněny.

4.4.1 Aproximace plochy nativní metodou

Stejně jako u křivek i zde byla zpočátku zvolena tato metoda z důvodu snadné implementace a nedostatečného teoretického základu umožňující realizaci jiných postupů. V prvním kroku tedy vezmu vstupní rovinu a vytyčím si základní měřící body. Stejně jako u křivek i zde počítám největší vzdálenost od skutečné plochy. Vytyčení bodů je následující. Kromě rohových bodů vyberu ještě body ležící uprostřed každé hrany ohraničující rovinu a bod, který leží přesně uprostřed dané roviny. Pro následující body tedy provedu výpočet jejich vzdálenosti od skutečné plochy. Opět upozorňuji, že výpočet probíhá v Eukleidovském nikoliv v parametrickém prostoru. Pokud jeden z vybraných bodů nesplňuje toleranci, pokračuji v dělení dle zadaného směru (viz. kapitola 4.4.2.1). V opačném případě musím provádět testování na další množině bodů. To probíhá podobně jako u křivek způsobem, kdy jsou hrany rozděleny na veliké množství malých úseků, které jsou postupně testovány. Jak je možné si z popisovaného postupu všimnout, tak opět zde nebudou pokryty všechny případy, a proto musel být i tento postup později

nahrazen.



Obrázek 16: Ukázka referenčních testovacích bodů (tečky) a náznak rozkrokování (čárky)

4.4.2 Aproximace plochy převodem na Beziérovy pláty

Stejně jako u křivek i zde můžeme aproximaci rozdělit na dva kroky:

- převod NURBS plochy na Beziérovy pláty
- dělení jednotlivých ploch na jemnější segmenty splňující svou plochostí toleranci

4.4.2.1 Převod NURBS plochy na Beziérovy pláty Jelikož lze Beziérovu plochu chápat jako tenzorový součin křivek, o kterém se můžete více dozvědět v [3], můžeme využít algoritmus pro převod NURBS křivky na křivky Beziérovy, s tím že se zde jedná o rovinu, tudíž bude potřeba celý algoritmus lehce modifikovat. Modifikace algoritmu spočívá především v převodu plochy na množinu křivek, buďto ve směru u nebo v. Jeden řádek řídících bodů plochy reprezentuje rovněž řídící body křivky. Dále pak musí tyto nově vzniklé křivky obsahovat uzlový vektor, který vybereme na základě zvoleného směru u nebo v. Získané křivky necháme převést na Beziérovy křivky. Z těchto křivek je poté potřeba vytvořit nové, které budou reprezentovat opačný směr oproti tomu původnímu. To provedu tak, že počáteční a poté následující body každé křiky jsou i řídícím body křivky opačného směru. Použitý uzlový vektor odpovídá požadovanému směru. Tyto nové křivky opět necháme převést na Bézierovy křivky dle výše zmiňovaného algoritmu. Výsledkem je množina Bézierových plátu, kterou budeme následně dělit algoritmem de Casteljau dokud nesplní danou toleranci. Tyto operace jsou realizovány v metodě `createSubdivision()`. Výsledný Beziérův plát má následující vlastnosti:

- stupeň plochy odpovídá stupni(m_1, m_2) NURBS plochy
- počet řídicích bodů je roven $(m_1 + 1) \times (m_2 + 1)$

4.4.2.2 Algoritmy pro následné dělení Beziérových plátů Jak jsem již zmiňoval výše, podstatná část algoritmu je obsažena v metodě `createSubdivision()`, kde na vstupu obdržím Beziérův plát. Na výstupu poté dostaneme množinu obdélníků, odpovídajících svou plochostí toleranci a pole stromových struktur, odpovídající vzniklým hranám a jejich potomkům. Práci této metody by šlo jednoduše popsat takto:

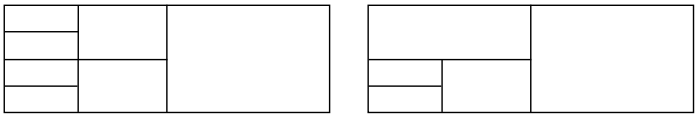
1. určení plochosti metodou `isFlat()`. Pokud splňuje, pak uložím rohové body do proměnné typu `Rectangle` a uložím do seznamu.
2. pokud nesplňuje pokračuj na bod 4
3. určím směr dělení, rozdělím plochu v daném směru na polovinu a nově vzniklé hrany uložím jako potomky do stromu nebo jako rodiče do seznamu.
4. rekurzivně přejdu na bod 1

Jako první popíšu metodu `isFlat()`, kde vstupem je Beziérův plát a výstupem informace o plochosti plochy a případná informace o tom, který směr nesplňuje toleranci ϵ. Tato metoda funguje v podstatě stejně jako testování tolerance u křivek s tím rozdílem, že musím počítat toleranci křivek jednak v u směru ale také ve v směru definované plochy. Pro určení zda splňuje toleranci či nikoliv opět využíváme znalostí konvexního obalu.

Nyní přejdeme k popisu bodu 2. Jak je zde vidět, tak tento bod se skládá ze dvou kroků. V zájmu zachování jednoduchosti budou tyto kroky popsány zvlášť. Začněme tedy prvním krokem. Když plocha neprojde testem na plochost, tak obdržíme informaci, o který směr se jedná. Pokud toleranci nesplňují oba směry, dělení probíhá střídavě pro jeden nebo druhý směr v závislosti na předchozím dělení. Pokud toleranci nesplňuje pouze jeden směr, pak následující dělení probíhá v tomto daném směru bez ohledu na předchozí dělení. Důvod proč dochází ke změně směru je ten, že kdybychom na základě plochosti testovali a současně dělili nejprve v jednom směru a následně ve druhém, tak výsledný počet obdélníků by byl větší obr. 17 a) než při střídavém dělení obr. 17 b), neboť toto dělení by zasáhlo i oblasti, které by střídavým dělením toleranci splňovaly o mnoho dříve, ale takto by byly zbytečně dále děleny. Výhodou tohoto řešení je, že získáme větší přesnost, ale současně vlivem většího počtu dojde ke zpomalení v dalších krocích a současně k nárustu počtu trojúhelníku, čímž by rovněž došlo ke zbytečnému zpomalení navazujících aplikací nebo dalších grafických programů, které by s výsledným objektem pracovaly, neboť sekundárním cílem tesselace je získat co nejmenš počet trojúhelníků při co nejmenším zkreslení.

Během dělení ovšem může nastat případ, kdy nově vzniklý obdélník leží kompletně mimo plochu definovanou ořezovými křivkami. Proto musí být před dalším zpracováním všechny obdélníky na tuto vlastnost otestovány. Původní vyhodnocování testu probíhalo pouze na základě kontroly, zda jsou všechny rohové body mimo rovinu definovanou křivkami či nikoliv. Ovšem můžou nastat případy, kdy jsou všechny čtyři body mimo polygon a přesto na ploše leží ořezové křivky. Proto je potřeba ještě provést test, zda nějaká část křivky v daném obdélníku neleží. K tomu, zda daný bod leží či neleží uvnitř polygonu, je využita proužková metoda, která se rychlostně i v rámci přesnosti nejvíce osvědčila. Ovšem vlivem zaokrouhlování není tato metoda nejpřesnější a i u ní

dochází k občasným chybám. Tato metoda je použita ve všech částech programu, kde je potřeba určit polohu bodů vůči obecnému polygonu.



(a) (b)

Obrázek 17: Rozdíl mezi dělením po jednotlivých směrech obr. a) oproti střídavému dělení obr. b)

Takovýmto dělením, kdy výsledkem jsou jsou obdélníky různých rozměrů, nám na výsledné ploše vznikají díry. Abychom tomuto jevu zabránili musíme si zapamatovat, kolik sousedních obdélníků hrana obsahuje. Na základě těchto informací je daná hrana dle počtu nalezených bodů dělena. Což je mnou zmiňovaná druhá část tohoto algoritmu.

K realizaci tohoto problému jsem využil datové struktury zvané „binární strom". Dále je potřeba si vytvořit pomocné objekty. První z nich `Edge` reprezentuje jednu z hran tvořeného obdélníku. Hodnota `edgeIndex` udává index hrany, který je uložen ve zvláštním poli jenž drží indexy všech vzniklých hran. `lchildIndex` a `rChildIndex` jsou indexy hran které vzniknou při dělení původní hrany na dvě poloviny.

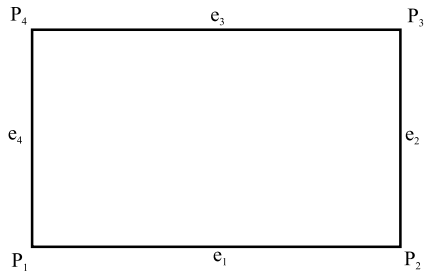
```
struckt  Edge
{
        Vector3 p1;
        Vector3 p2;
        size_t  edgeIndex;
        size_t  lChildIndex;
        size_t  rChildIndex;
}
```

Výpis 9: Definice struktury `Edge`

Proměnná `EdgeRactangle` reprezentuje vytvářené dělené obdélníky. Primárně neobsahuje informaci o jeho rohových bodech, ale představuje hranovou reprezentaci daného obdélníku. Při závěrečném kroku tohoto algoritmu je z této datové struktury odvozen datový typ `Rectangle`.

```
struct  EdgeRactangle
{
        Edge e1;
        Edge e2;
        Edge e3;
        Edge e4;
}
```

Výpis 10: Definice struktury `EdgeRactangle`

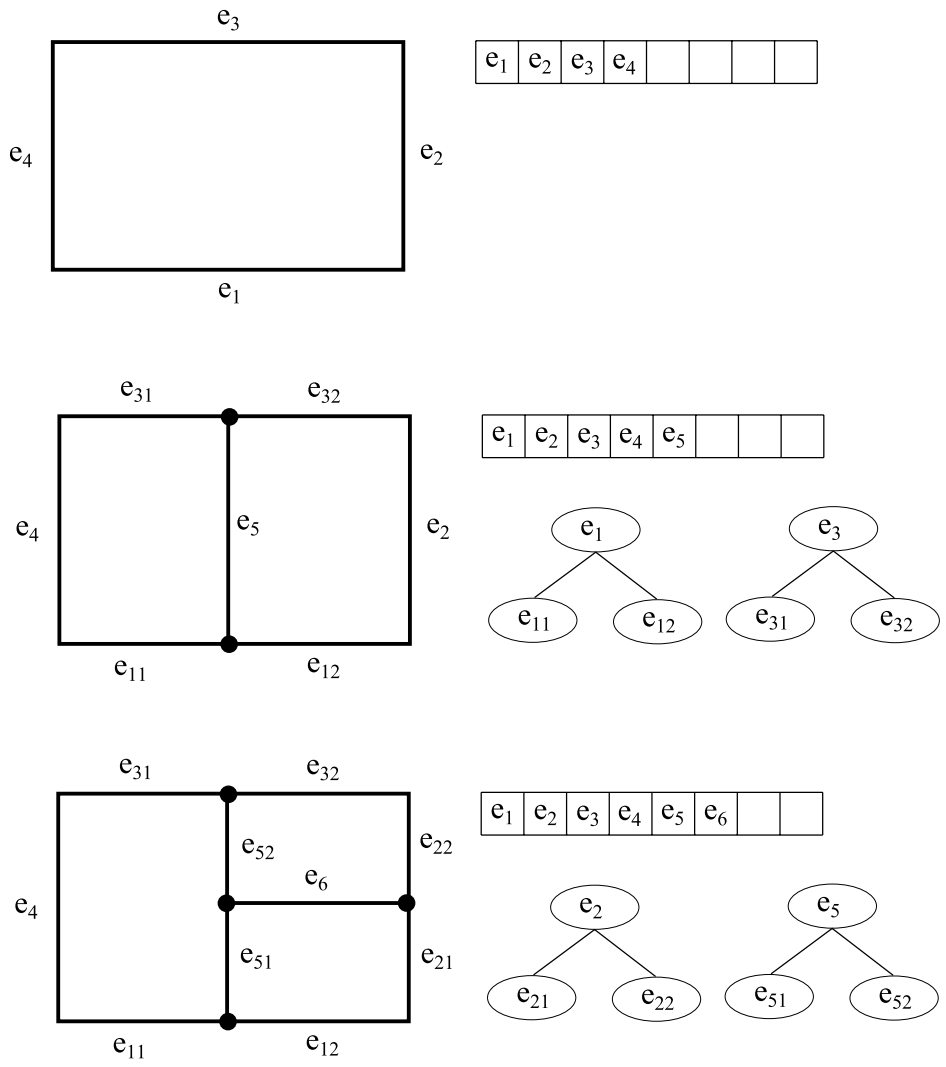


Obrázek 18: Hranová reprezentace obdélníku

Jak tedy postupovat? Ze zásobníku vyjmeme jeden Bézierův plát. Na základě jeho rohových řídících bodů vytvořím první obdélník datového typu `EdgeRactangle`. Každá nově vytvořena hrana je rovněž kořenem stromu, který bude uchovávat své potomky, jenž vznikají dělením dané hrany. V našem případě jsou tak vytvořeny 4 kořenové uzly. Současně je vytvořeno pomocné pole `EdgeTreeArray`, které uchovává indexy všech vzniklých hran.

Po této přípravě můžeme následně zavolat metodu `createSubdivision()`. Kde po vykonání výše zmiňované části, pokračujeme ve vytváření potomků rodičovského obdélníku. Nové obdélníky vznikají na základě dělení původních nebo nově vzniklých osových hran. Pokud hrana vznikla dělením nějaké hrany, pak její index musí být uložen jako levý nebo pravý potomek v původní hraně. Jestliže vznikne hrana nová, pak se nejedná o potomka, ale o kořen nového stromu. Pro nově vzniklé obdélníky vždy rekurzivně volám tuto metodu, dokud metoda `isFlat()` nevrací hodnotu $true$ pro všechny vzniklé obdélníky.

Provedením metody `createSubdivision()`, získáme množinu stromů, reprezentující hrany a jejich potomky a pole indexů všech vzniklých hran. Nyní tyto údaje musíme převést na finální obdélníky, které vedle svých rohových bodů, obsahují i hranové body, které reprezentují rohový bod sousedního obdélníku. Metoda, pro tento převod určená, se jmenuje `setNeighboursPoints()` a pracuje následujícím způsobem. Tato metoda na vstupu očekává dělením získané obdélníky typu `EdgeRactangle` a pole hran `EdgeRactangles`. Pro každý obdélník ze vstupní množiny testuji zda každá hrana vstupního obdélníka obsahuje či neobsahuje levého nebo pravého potomka. Pokud ne pak hrana nemá žádné sousedy, jejichž rohové body by ležely na zkoumané hraně (samozřejmě mimo krajní body hrany) a můžu tedy pokračovat v průzkumu další hrany respektive dalšího obdélníku. Pokud ovšem hrana potomka či potomky obsahuje, tak je volána metoda `passageOfTree()`, kde bude probíhat další rekurzivní zkoumaní. Průzkum zde probíhá podobně jako v předchozí metodě s tím rozdílem, že pokud je levý nebo pravý potomek `NULL` , pak na základě toho, o který směr se jedná, určíme zda budeme ukládat levý nebo pravý bod zkoumané hrany. Rekurze končí, pokud rodič neobsahuje levého ani pravého potomka. Nyní už mám vše potřebné pro to, abych mohl začít triangulovat.



Obrázek 19: Dělení hran a vznik nových stromů

5 Triangulace

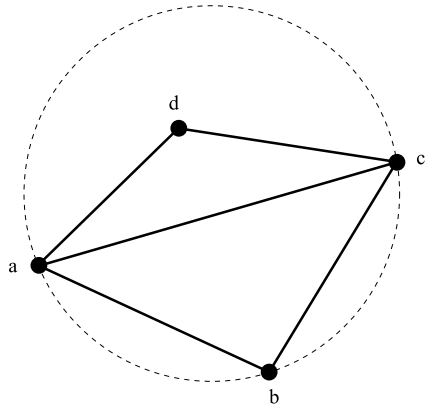
Dalším krokem v tesselaci je triangulace naaproximované a ořezané plochy. Co je to triangulace a její důležité vlastnosti jsou popsány v následujících definicích.

Definice 5.1 *Maximální planární rozklad je rozklad indukovaný jistou množinou vrcholů tak, že žádná hrana spojující dva vrcholy nemůže být přidána aniž by byla porušena planarita.*

Definice 5.2 *Triangulace P je maximální planární rozklad, jehož množina vrcholů je P.*

Věta 5.1 *Nechť P je množina obsahující n bodů v rovině a nechť k označuje počet těch bodů z P, které leží na konvexním obalu P. Pak kterákoliv triangulace P má právě $2n - 2 - k$ trojúhelníků a $3n - 3 - k$ hran.*

Věta 5.2 *(Existence ilegální hrany): Nechť ac je hrana, která inciduje s trojúhelníky abc a acd a nechť C je kružnice procházející body a, b, c. Hrana ac je ilegální právě tehdy, kdy d leží uvnitř kružnice viz. obrázek 20.*



Obrázek 20: Ukázka ilegální hrany

Nyní po krátkém teoretickém úvodu, kde jsem definoval potřebné základní vlastnosti triangulace, můžeme pokračovat v popisu implementace v mém programu. Během samotného popisu doplním ještě několik zajímavých informací o dalších možných přístupech.

5.1 Ořez plochy ořezovými křivkami

Před tím než dojde k triangulaci musíme ještě provést ořezání naaproximované plochy ořezovými křivkami. Jak tedy postupovat. Ořez se v mém případě neprovádí najednou pro celou plochu, ale zvlášť pro každý obdélník. V prvních verzích programu se k tomu využíval jednoduchý postup testování zda nějaký bod aproximačního polygonu, reprezentujícího křivku, leží uvnitř obdélníku či nikoliv. Pokud ano vypočetl se k daným

úsečkám průsečík s hranami obdélníku. Toto se provádělo jak pro vnitřní tak i vnější ořezové křivky. Problém tohoto postupu byl ten, že nemusel pokrýt všechny možné varianty. Příkladem budiž varianta, kdy žádný z bodů křivky neleží uvnitř vstupního obdélníku, ale přesto její část jím prochází. Aby se těmto problémům zamezilo, byl využit algoritmus popisovaný v [7]. Tento algoritmus pracuje na podobném principu jako Z-buffer, a to tak že daná rovina je překryta ořezovými křivkami a ty, jež jsou její součástí ať už průsečíkově či samotným vlastním bodem, jsou uloženy do pomocného pole, reprezentovaného všemi body, které budou v následujícím kroku triangulovány. Toto pole samozřejmě obsahuje i body, které jsou součástí vstupního obdélníku, které leží uvnitř křivky.

Jelikož na triangulaci pošleme, jen část (jednu aproximační rovinu) a nikoliv celou plochu, musíme provést před vydáním množiny bodů na triangulaci, ještě jednoduchý test, který nám umožní vyhnout se poměrně složitému procesu triangulace. Tento test vznikl na základě pozorování, kdy bylo zjištěno že většina obdélníku přicházejících na vstup, jsou jen ryzí obdélníky tvořeny čtyřmi rohovými body. Jelikož je pro tento případ triangulace známá, stačí vytvořit ze čtyř rohových bodů dva trojúhelníky a vyhnout se tak algoritmu inkrementálního vkládání popisovaném v kapitole 5.2.3.1. Přestože na vstup mohou přijít jen čtyři body, nemusí se vždy jednat o obdélníkové těleso. Původně test probíhal jen na ověření, zda toto čtyřhranné těleso obsahuje uhlopříčky stejné délky. Později však byl objeven případ, kdy tento test nebyl dostatečný, a proto se muselo přejít na testování svíraných uhlů danými navazujícími hranami, kde všechny musely svírat úhel 90°.

5.2 Metody využívané k triangulacii

Během realizace této práce jsme narazil na tři způsoby, jak lze množinu bodů natriangulovat. Jedná se o

- Greedy triangulaci
- triangulace pomocí Voronoiova diagramu
- Delanayeho triangulaci

V následujících několika kapitolách se budu snažit alespoň stručně popsat první dvě zmiňované a detailněji poslední Delanayeho, neboť ta je využita i v mém programu.

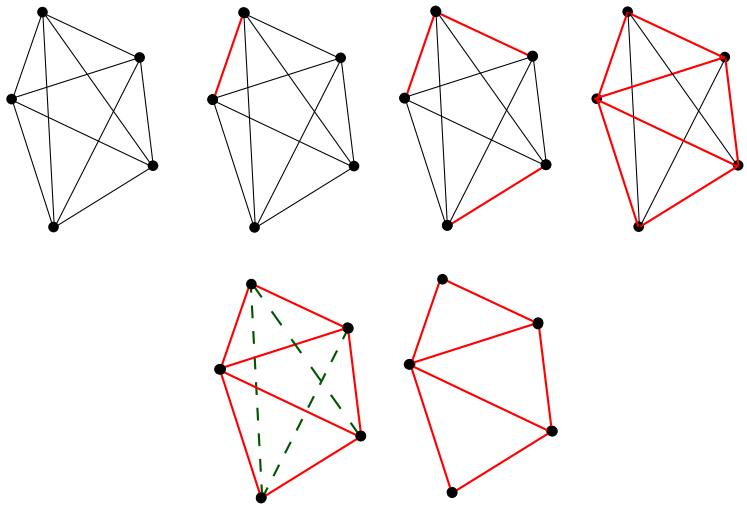
5.2.1 Greedy triangulace

Algoriitmus popisovaný v [15] je výhodný především z hlediska jednoduché implementace. Algoritmus pracuje následujícím způsobem:

1. z množiny vstupních bodů vygeneruji všechny možné hrany
2. vygenerované hrany seřadím vzestupně podle délky

3. vyberu nejkratší nevloženou hranu a za předpokladu, že se neprotíná s žádnou jinou již vloženou, ji vložím do triangulace

4. opakuji krok 3. dokud nezbývá poslední hrana, nebo celkový počet hran v triangulaci je menší než $3n - 6$



Obrázek 21: Ukázka greedy triangulace

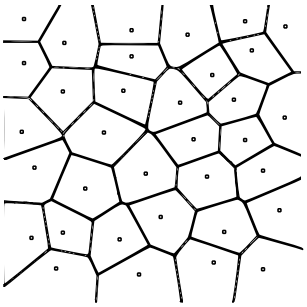
Nevýhodu tohoto algoritmu je jeho vysoká složitost $O(n^3)$, kterou lze optimalizovat na $O(n^2 \log(n))$. Další nevýhodou je, že výstupní síť může obsahovat tvarově nevhodné trojúhelníky. Z těchto důvodů tato metoda není často využívána.

5.2.2 Voronoiův diagram

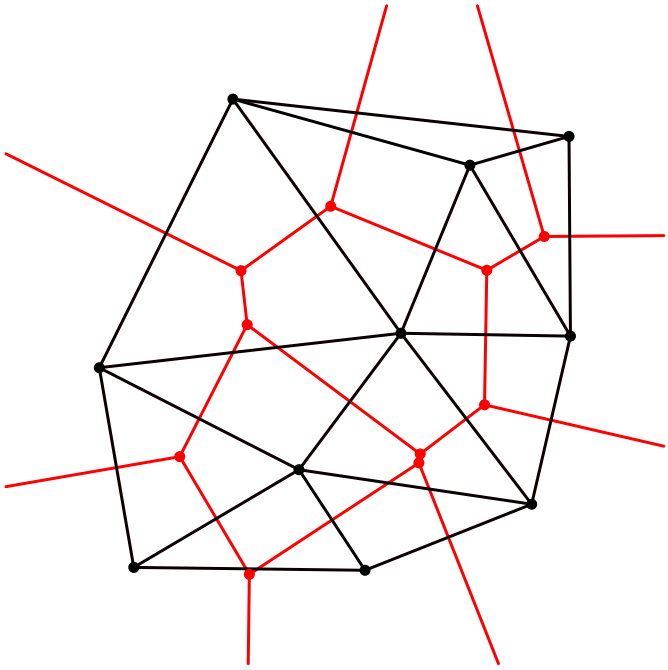
Další možností, jak natriangulovat množinu bodů, je Voronoiův diagram. Dle [18] je definován takto.

Definice 5.3 *Voronoiův diagram $V(P)$ nad množinou bodů $P = \{p_1, p_2, \ldots, p_n\}$ v rovině představuje rozklad množiny P na n uzavřených či otevřených oblastí $V(P) = \{V(p_1), V(p_2), \ldots, V(p_n)\}$ takových, že každý bod $q \in V(p_i)$ je blíže k bodu pi než k jakémukoliv bodu $p_j \in P$. Uzavřená buňka $V(p_i)$ se nazývá Voronoiva buňka (polygon). Pro libovolný bod $q \in V(p_i)$ a libovolnou buňku $V(p_i)$ platí $d(q, p_i) \leq d(q, p_j)$.*

Všechny body oblasti $V(p_i)$ mají stejného nejbližšího souseda. Duálním grafem k Voronoiovu diagramu je Delaunay triangulace. Body p_i tvoří současně vrcholy Delaunay triangulace současně body p_i, p_j tvoří hranu t v Delaunay triangulaci právě když, $V(p_i)V(p_j)$ sdílejí společnou Voronoiovu hranu. Středy kružnic opsaných trojúhelníkům v Delaunay triangulaci představují vrcholy Voronoiova diagramu.



Obrázek 22: Ukázka Voronoiova diagramu



Obrázek 23: Využití Voronoiva diagramu (červeně) při triangulaci (černě)

5.2.3 Delaunayho triangulace

Nechť P je množina bodů a Vor(P) je Voronoiův diagram P. V(P) nechť označuje Voronoiův polygon pro bod $p \in P$. K Vor(P) sestrojíme duální graf Del(P) takto: Do každého bodu množiny P umístíme uzel grafu Del(P). Jestliže V(p) a V(q) ($p, q \in P$) mají společnou hranu, pak uzly p, q v Del(P) spojíme hranou, která je úsečkou.

Věta 5.3 *Delonayho graf množiny bodů v rovině je planární graf (tj. Hrany sestrojené podle předchozího postupu se neprotínají).*

Věta 5.4 *Teorém: Nechť P je množina bodů v rovině. Triangulace T množiny P je legální právě tehdy když T je Deloneho Triangulace množiny P.*

Algoritmy pro pro Delaunayho triangulaci

1. inkrementální vkládání
2. algoritmus radikálního zametání
3. rozděl a panuj
4. metoda lokálního zlepšování
5. převod do vyšší dimenze

Pro účely mé práce byla využita první metoda inkrementálního vkládání, která bude posléze detailněji popsána. Stručný popis ostatních metod lze nalézt v [10].

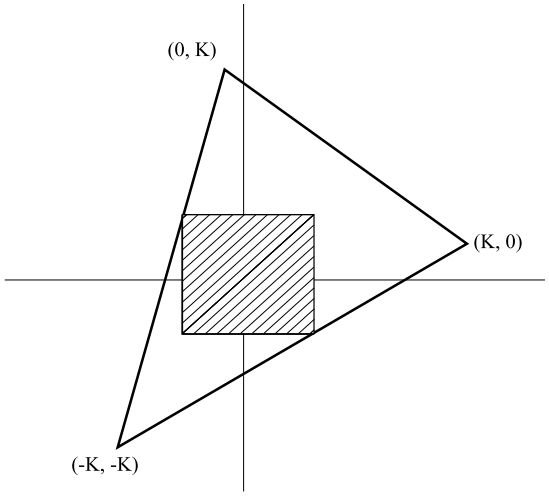
5.2.3.1 Metoda inkrementálního vkládání Jedná se o online algoritmus tj. vstupní množinu bodů není nutné dopředu znát a nové body mohou být libovolně dle potřeby vkládány. Další výhodou a důvodem využití v mé práci je jednoduchá implementace.

Algoritmus lze rozdělit do čtyř fází:

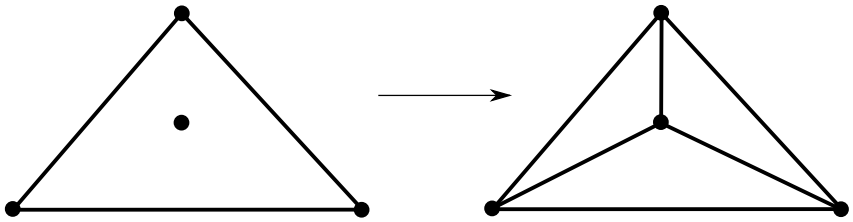
- konstrukce obalového trojúhelníku (simplexu)
- přidání bodu do triangulace
- legalizace triangulace
- odstranění hran obalového trojúhelníku

V prvním kroku tedy vytvoříme obalový trojúhelník z uměle vložených bodů, který je dostatečně veliký tak, aby obsahoval všechny příchozí body. Abych se při konstrukci vyhnul různým problémům vzniklých pevným vložením umělých bodů, využil jsem proto postup, který je popisován v [10]. Souřadnice vrcholu trojúhelníka si odvodím od $\min - \max$ boxu. Což jsou v mém případě souřadnice $(0, K)$ $(K,0)$ $(-K,-K)$ viz. obrázek 24, kde K je desetinásobek velikosti $\min - \max$ boxu.

Nyní, když máme vytvořen obalový trojúhelník, můžeme přistoupit k inkrementálnímu vkládání jednotlivých bodů. Vložím tedy první bod a trojúhelník se mi rozdělí na tři části viz obrázek 25. Nově vzniklé trojúhelníky a jejich vrcholy pomocí metody `addTriangle()` uložím. Všechny trojúhelníky ukládám do pomocného pole a samotné vrcholy do grafové struktury typu DAG s tím, že graf je totožný s vytvářenou trojúhelníkovou sítí. Toto uspořádání je nutné pro rychlé vyhledávání trojúhelníku, obsahujících nově vložený bod. Aby bylo zaručeno bezproblémové testování ilegálnosti hran, je potřeba uchovávat i správné pořadí vrcholů proti směru hodinových ručiček. Další možnou variantou ukládání vrcholu je metoda zjemňování triangulace, kterou jsem ovšem v době implementace tohoto algoritmu neznal a tudíž ani neuvažoval nad jejím užitím.



Obrázek 24: Konstrukce obalového trojúhelníka v závislosti na velikost $\min - \max$ boxu [10]



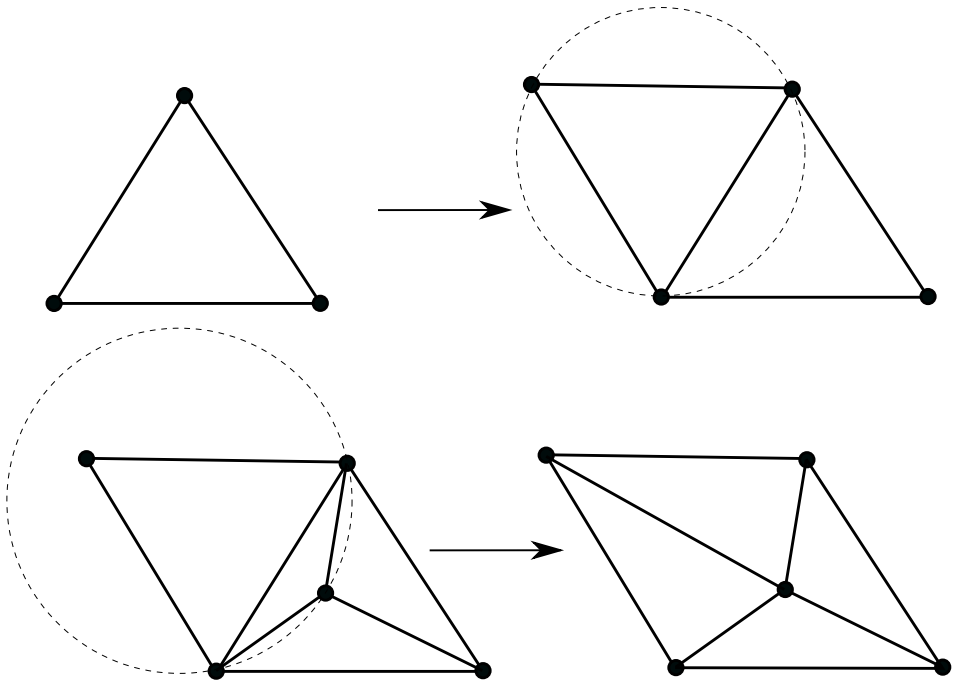
Obrázek 25: Ukázka vložení nového bodui

Po každém novém vložení, musím provést test na ilegální hrany. Vezmu tedy hrany nově vzniklého trojúhelníka otestuji je pomoci determinantu, jenž je odvozen v [9] a který má následující tvar

$$\det = \begin{vmatrix} a_x - d_x & a_y - d_y & (a_x - d_x)^2 + (a_y - d_y)^2 \\ b_x - d_x & b_y - d_y & (b_x - d_x)^2 + (b_y - d_y)^2 \\ c_x - d_x & c_y - d_y & (c_x - d_x)^2 + (c_y - d_y)^2 \end{vmatrix} . \tag{16}$$

Jedná se zde o test zda nějaký bod leží uvnitř kružnice, která opisuje nově vzniklý trojúhelník. Pokud žádné body v daných kružnicích neleží $(\det \leq 0)$, nejedná se o ilegální hrany a my tak můžeme pokračovat ve vkládání dalšího bodu. V opačném případě $(\det > 0)$ musí být inkriminované hrana prohozena. Tím, ale můžou vzniknout nové ilegální hrany, které musí být opět stejným způsobem jako v předchozím případě rekurzivně otestovány, dokud všechny hrany nenabydou legálnosti. Ještě dodám, že při výpočtu je nutná správná orientace bodů.

Po vložení všech bodů nám již zbývá odstranit, trojúhelníky jejichž vrchol je součástí jednoho ze tří vrcholů obalového trojúhelníka. Tím získáme výslednou triangulaci, která nezohledňuje ořezy ani výřezy a některé důležité hrany.



Obrázek 26: testování ilegálnosti hran

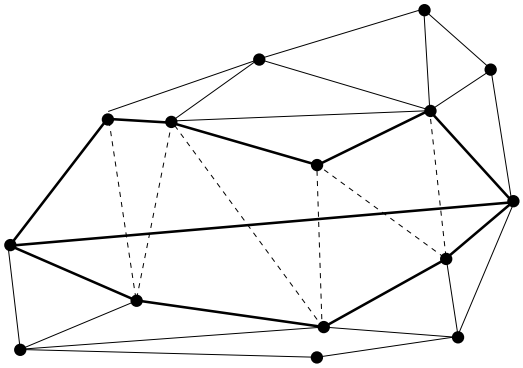
5.2.4 Anglada

Jak již víme, tak předchozí metoda inkrementálního vkládání nám vytvoří trojúhelníkovou síť s konvexním obalem a to i přesto, že je množina vstupních bodů definována jako nekonvexní polygon, nebo že triangulovaná plocha obsahuje díry, definované ořezovými křivkami. Abychom tedy získali polygon požadovaného tvaru, musíme mu dané hrany vnutit. K tomuto účelu slouží algoritmus zvaný „Anglada“.

Algoritmus, který je popisován v [9], je aplikován v metodě `anglada()`, kde vstupem je množina trojúhelníků a dvěma body zadaná vnucená hrana. Výstupem je opět množina trojúhelníků, zohledňující jak vnucené hrany tak i možné vnitřní díry. Jedná se o algoritmus, který umožňuje vkládaní jak bodů tak i hran. Pro můj případ stačila ovšem pouze varianta s vkládáním hran. Jelikož postup je detailně popsán v [9], vystihnu jen nejdůležitější kroky. Na vstupu tedy dostanu dva body, reprezentující vnucenou hranu $\mathbf{P}_a\mathbf{P}_b$. Pro ní nalezneme všechny hrany, které se s ní protínají. Obrázek 27 znázorňuje ukázku vnucené hrany a jí zasažené oblasti.

Jak je dále vidět z obrázku 27, tak vnucená hrana nám danou oblast rozdělí na horní a dolní část, pro kterou jsou vytvořeny dva seznamy (`nPointH` pro horní oblast a `nPointD` pro dolní), které obsahují indexy vrcholů rušených trojúhelníku. O to zda má být nalezený vrchol přiřazen do dolní nebo horní oblasti, se stará následující determinant

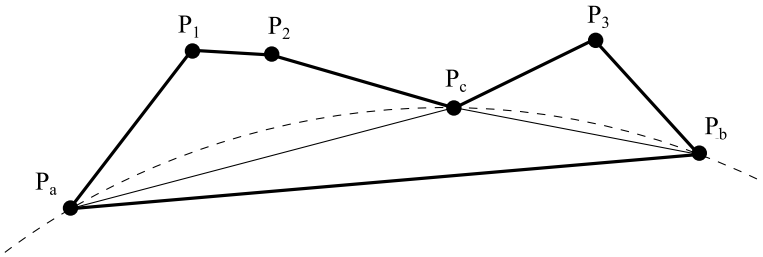
$$\det = \begin{vmatrix} v_x & v_y \\ u_x & u_y \end{vmatrix}, \tag{17}$$



Obrázek 27: Vnucení hrany ab. Protnuté hrany budou posléze vymazány [9]

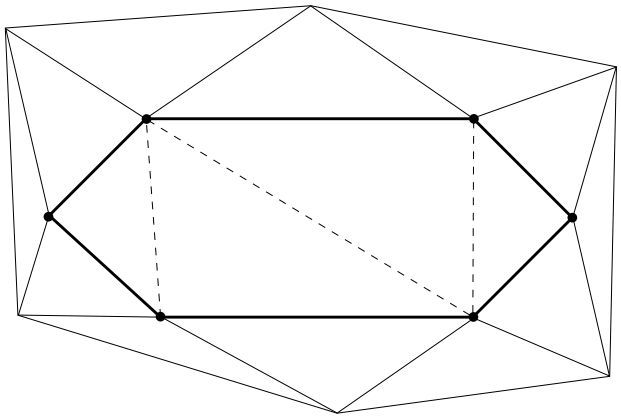
kde $\mathbf{u} = \mathbf{P}_b - \mathbf{P}_a$ a $\mathbf{v} = \mathbf{P}_c - \mathbf{P}_a$ viz obrázek 28. Pokud výsledek determinantu vychází kladný potom vektor $\mathbf{v}$ směřuje doprava (nahoru) od vektoru $\mathbf{u}$ a naopak. Pokud je determinant roven nule, pak oba vektory leží ve stejném směru. Nyní, kdy mám body rozděleny na horní a dolní oblast, potřebuji vytvořit novou triangulaci pro vnucenou hranu. Dle [8] můžeme triangulovat, jen zasaženou oblast, bez toho abychom mimo ní vytvořili ilegální hrany. Triangulace zasažené oblasti se provádí zvlášť pro horní a dolní oblast. Toto umožňuje námi vnucená hrana, neboť obě oblasti se přes ní vzájemně nevidí. Funkce algoritmu je shodná jak pro horní tak dolní oblast. Proto stačí popsat pouze jednu stranu v mém případě tu horní.

V prvním kroku procházím seznam `nPointH` a spolu s hranou $\mathbf{P}_a\mathbf{P}_b$ vytvořím trojúhelník, který posléze otestuji na to, zda kružnice, která opisuje daný trojúhelník, neobsahuje žádný bod z daného seznamu. Využiji stejného determinantu jako v případě inkremetálního vkládání. Pokud ano, pak pokračuji v seznamu dále a vytvářím nové trojúhelníky do té doby dokud alespoň jeden neprojde testem viz. obrázek 28. Hrany takto nově vzniklého trojúhelníku nám danou oblast rozdělí na další dvě oblasti, kde jedna bude obsahovat body $\mathbf{P}_1$, $\mathbf{P}_2$ a druhá bod $\mathbf{P}_3$. Tyto oblasti dále rekurzivně dělím, dokud pro všechny vstupní body není vytvořen trojúhelník. Tento postup je aplikován v metodě `evaulateIntersectTriangles()`.

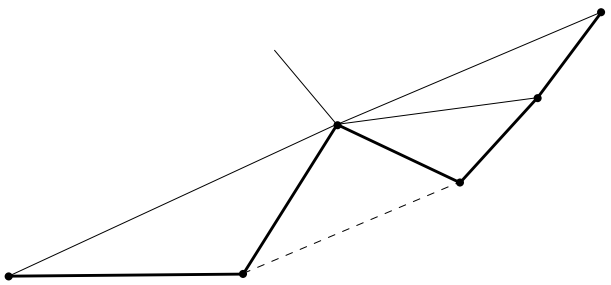


Obrázek 28: Ukázka nalezení optimálního trojúhelníku a rozdělení oblasti na nové dvě oblasti [9]

Vnucením hran jsme, ale nevyřešili problém nadbytečných trojúhelníků vyplňující díry a okraje polygonu. Řešení problému je zde opět poměrně jednoduché. Lze totiž využít té vlastnosti, že všechny tři vrcholy trojúhelníků, které vyplňují díru, leží na některém z bodů vnucené hrany viz. obrázek 29, dále je potřeba vzít bod, který leží na úsečce a otestovat zda leží či neleží uvnitř polygonu nebo zda leží na vnucené hraně. To samé platí i o trojúhelnících na okraji polygonu, které svou přítomností vytváří z nekonvexního polygonu polygon konvexní viz. obrázek 30.



Obrázek 29: Nadbytečné trojúhelníky, které vyplňují díru



Obrázek 30: Trojúhelník rušící nekonvexitu polygonu.

6 Výpočet normálového vektoru NURBS plochy

Posledním krokem naší tesselace je výpočet normály pro jednotlivé vrcholy trojúhelníků. Vypočíst ji můžeme dvěma způsoby:

- klasickou metodou
- parciální derivaci bázových funkcí

6.1 Výpočet normálového vektoru klasickou metodou

Tuto metodu jsem nazval klasickou, neboť využívá klasických postupů používaných pro výpočet normálových vektorů v rovině. V našem případě je lehce modifikovaná pro NURBS plochy.

Na vstupu získáme koordináty u a v, odpovídající parametrickým souřadnicím plochy. K těmto koordinátům přičteme hodnotu 0.01 a získáme hodnoty v_1 a u_1. Zde si ale musíme dát pozor zda nepřekročíme rozsah parametru plochy pro daný koordinát, pokud ano místo přičítání hodnotu 0.01 odečteme. Takto vzniklé parametry dosadíme do vzorce definující NURBS plochu a získáme body $\mathbf{P}_c, \mathbf{P}_1, \mathbf{P}_2$,

$$\mathbf{P}_c = \mathbf{S}(u, v), \mathbf{P}_1 = \mathbf{S}(u_1, v), \mathbf{P}_2 = \mathbf{S}(u, v_1)\,,$$

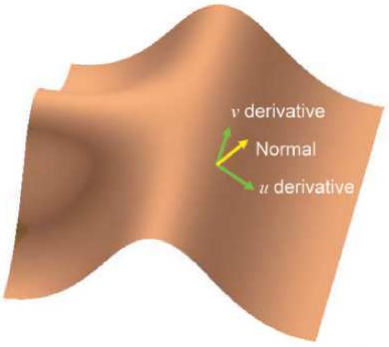
pomocí kterých vypočteme směrové vektory $\mathbf{u}_1$, $\mathbf{u}_2$

$$\mathbf{u}_1 = \mathbf{P}_c - \mathbf{P}_1\,, \mathbf{u}_2 = \mathbf{P}_c - \mathbf{P}_2\,,$$
$$\mathbf{n} = \mathbf{u}_1 \times \mathbf{u}_2\,. \tag{18}$$

V případě, kdy je vstupní parametr na hranici plochy a je potřeba od něj odečítat, musíme $\mathbf{P}_c$ a $\mathbf{P}_1$ respektive $\mathbf{P}_2$ prohodit. Jinak by výsledná normála byla opačného směru a na výsledku by se to projevilo černou plochou. Nyní, když známe oba směrové vektory, stačí vypočíst vektorový součin a získat tak požadovaný normálový vektor. Tento způsob výpočtu přinesl ve výsledku viditelné nepřesnosti při zobrazení křivějších ploch. Proto byla tato metoda nahrazená metodou využívající parciální derivaci bázových funkcí.

6.2 Výpočet normály pomocí derivace bázových funkcí

Základem výpočtu je tedy parciální derivace plochy $\mathbf{S}(u, v)$, která odpovídá směrnicím tečen ve směru jednotlivých souřadnicových os. V našem případě se jedná o u a v.



Obrázek 31: Výpočet normály z u a v pomocí parciální derivace [12].

Zavedu tedy parciální derivaci s následujícími pomocnými funkcemi

$$\mathbf{A}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u)N_j^q(v)\mathbf{P}_{i,j}w_{i,j}\,, \tag{19}$$

kde $\mathbf{A}(u,v)$ se nazývá vektorová funkce a jeji první derivace je následující

$$\frac{\partial}{\partial u}\mathbf{A}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \frac{\mathrm{d}}{\mathrm{d}u}N_i^p(u)N_j^q(v)\mathbf{P}_{i,j}w_{i,j}\,. \tag{20}$$

Dále si zavedeme funkci $\mathbf{w}(u,v)$, která reprezentuje homogení souřadnici a má následující tvar

$$\mathbf{w}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u)N_j^q(v)w_{i,j}\,. \tag{21}$$

Její výslednou derivaci poté zapíšeme takto

$$\frac{\partial}{\partial u}\mathbf{w}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \frac{\mathrm{d}}{\mathrm{d}u}N_{i,p}(u)N_{j,q}(v)w_{i,j}\,. \tag{22}$$

Výsledné pomocné funkce můžeme dosadit do rovnice

$$\frac{\partial}{\partial u}\mathbf{S}(u,v) = \frac{\frac{\partial}{\partial u}\mathbf{A}(u,v)\mathbf{w}(u,v) - \mathbf{A}(u,v)\frac{\partial}{\partial u}\mathbf{w}(u,v)}{\mathbf{w}(u,v)^2} \tag{23}$$

a získat tak směrový vektor ve směru u. Podobně, jako u vzorců 20 a 22, odvodíme $\frac{\mathrm{d}}{\mathrm{d}v}\mathbf{w}(u,v)$ a $\frac{\mathrm{d}}{\mathrm{d}v}\mathbf{A}(u,v)$ a dosadíme do vzorce

$$\frac{\partial}{\partial v}\mathbf{S}(u,v) = \frac{\frac{\partial}{\partial v}\mathbf{A}(u,v)\mathbf{w}(u,v) - \mathbf{A}(u,v)\frac{\partial}{\partial v}\mathbf{w}(u,v)}{\mathbf{w}(u,v)^2}\,, \tag{24}$$

čímž získáme i směrový vektor pro směr v. Z uvedených vzorců vyplývá, že ke správnému výsledku potřebujeme znát ještě první derivaci bázových funkcí. Pro tuto

derivaci lze využít jeden ze dvou vzorců. V [11] se k potřebné derivaci dojde pomocí koeficientů $C_{i,p,k}(u)$.

$$\begin{aligned}
C_{i,0,0}(u) &= N_i^0(u) \\
C_{i,p,0}(u) &= \frac{u_{i+p+1}C_{i+1,p-1,0}(u)}{u_{i+p+1}-u_{i+1}} - \frac{u_i C_{i,p-1,0}(u)}{u_{i+p}-u_i} \\
C_{i,p,0}(u) &= \frac{u_{i+p+1}C_{i+1,p-1,0}(u)}{u_{i+p+1}-u_{i+1}} - \frac{u_i C_{i,p-1,0}(u)}{u_{i+p}-u_i} \\
C_{i,p,k}(u) &= \frac{C_{i,p-1,k-1}(u) - u_i C_{i,p-1,k}(u)}{u_{i+p}-u_i} - \frac{C_{i+1,p-1,k-1}(u) - u_{i+p+1}C_{i+1,p-1,k}(u)}{u_{i+p+1}-u_{i+1}} \\
&\qquad\qquad \text{pro } 0 < k < p\,,
\end{aligned} \tag{25}$$

kde výslednou derivaci poté vypočteme následujícím způsobem.

$$\frac{\mathrm{d}}{\mathrm{d}u}N_i^p(u) = \sum_{k=1}^{p} p\; C_{i,p,k}(u)u^{k-1}\,. \tag{26}$$

Tento přístup je poměrně složitý a náročný na implementaci, kde musíme pro výpočet koeficientů vyšších řádů znát koeficienty řádů nižších a tudíž musí výpočet probíhat rekurzivně pro každý koeficient zvlášť (tento přístup navíc vede k vyšší složitosti). Druhou možností je uložení v trojrozměrném poli, které je poměrně nepřehledné co se orientace týče. O poznání jednodušší způsob jsem nalezl v [4], kde je požadovaná derivace odvozena takto

$$\frac{\mathrm{d}}{\mathrm{d}u}N_i^p(u) = \frac{p}{u_{i+p}-u_i}N_i^{p-1}(u) - \frac{p}{u_{i+p+1}-u_{i+1}}N_{i+1}^{p-1}(u)\,. \tag{27}$$

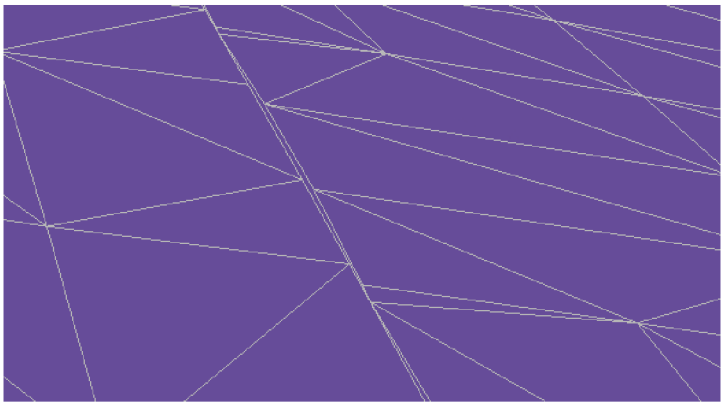
Nyní když známe obě směrnice, stačí jen vektorovým součinem a následně normalizací vypočíst požadovaný normálový vektor

$$\mathbf{t_u} = \frac{\partial}{\partial u}\mathbf{S}(u,v) \quad \mathbf{t_v} = \frac{\partial}{\partial v}\mathbf{S}(u,v) \tag{28}$$

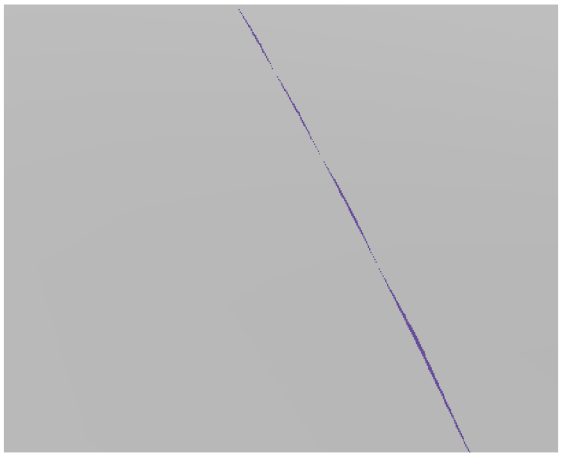
$$\mathbf{n} = \mathbf{t_u} \times \mathbf{t_v}\,. \tag{29}$$

7 Sewing algoritmus

Nyní již můj program umí natesselovat libovolnou NURBS plochu. Problém ovšem nastane v okamžiku, kdy se námi tesselovaný objekt skládá z více než jedné plochy. Tyto případy vlivem aproximace ořezových křivek vyvolávají nepřesnost, která na výsledném objektu způsobuje viditelné díry (cracky). Celá tato kapitola se proto bude věnovat řešení tohoto problému.



Obrázek 32: Ukázka díry na síťovém modelu



Obrázek 33: Ukázka díry na výsledném objektu

Jako inspiraci k dané problematice jsem využil algoritmus popisovaný v [6], který ovšem nepopisuje řešení případů, které se celkem běžně vyskytují a způsobují tak nefunkčnost daného řešení. O těchto problémech se zmíním v průběhu kapitoly. Samotný algoritmus bych rozdělil na 3 částí

- nalezení sousedních ploch pomocí BBH algoritmu

- nalezení dvojice sousedních bodů (každý z jedné plochy), jejichž vzájemná vzdálenost odpovídá uživateli zadané toleranci
- reparametrizace bodů a vygenerování nových trojúhelníků a normál

Vstupem algoritmu je uživatelem zadaná tolerance. Využívám stejné hodnoty jako je zadaná hodnota pro chybu aproximace. Experimentálně bylo ovšem zjištěno, že takto braná vzdálenost je nedostatečná, a proto je vždy vynásobena 2. Výstupem je sešitý objekt, neobsahující žádné díry. K maximálnímu zjednodušení práce byla potřeba nadefinovat následující pomocné struktury,

```
struct Vertex
{
        size_t m;
        size_t bId;
        Vector3 coord;
        bool orig;
        float distance;
        Vertex *v;
};
```

Výpis 11: Definice struktury `Vertex`

kde `m` je index plochy, které daný bod náleží, `coord` je souřadnice daného bodu, `bId` ID úsečky, které daný bod náleží. Jelikož jeden bod se vyskytuje ve dvou hraničních úsečkách, bere se proto hodnota té úsečky, jejíž bod je počáteční ve směru hodinových ručiček. Parametr `orig` nám říká zda je bod originální, tj. tvoří původní úsečku, nebo je uměle vytvořen na základě nejbližší vzdálenosti. Poslední parametr `v` je pointer nejblíže nalezeného bodu. Druhou používanou strukturou je struktura `Boundary`], která je definovaná takto,

```
struct Boundary
{
        size_t_ID mId;
        size_t bId;
        size_t tId;
        std::vector<Vertex *> vert;
};
```

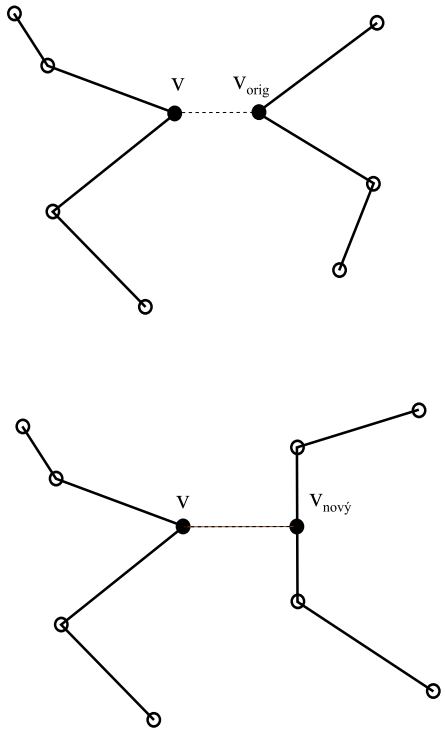
Výpis 12: Definice struktury `Boundary`

kde první dva parametry jsou shodné s předchozí definicí struktury `Vertex`. Proměnná `tId`, představuje informaci o tom, kterému trojúhelníku daná hraniční úsečka náleží. Seznam bodů, který je reprezentován parametrem `vert`, obsahuje všechny jak originální tak nově vytvořené body, které dané úsečce náleží.

7.1 Nalezení vhodné dvojice bodů

Cílem je nalezení bodu, který leží ,nebo bude vytvořen na sousední ploše a vytvoří tak dvojici s bodem protilehlým. Jako první krok šicího algoritmu je nalezení sousedních ploch, které svou vzdálenosti odpovídají zadané toleranci. K tomuto účelu byl využit již ve VRUTu vytvořený modul, který nalezne vhodné sousední plochy pomocí BVH algoritmu.

V zásadě je postup velice jednoduchý. Vezmu všechny body tvořící hranu dané plochy a postupně k ním hledám souseda. Sousedem je myšlen buď to bod z množiny bodů, tvořící hranu sousední plochy a nebo nejbližší vzdálenost ve vztahu bod hrana, která nám na inkriminované hraně vytvoří nový bod. Popsané případy jsou znázorněny na obrázku 34



Obrázek 34: Nalezení sousedních bodů. Nahoře nalezení originálního bodu. Dole nalezení neoriginálního bodu [6]

Jistě si musíte pomyslet, že nalezení hrany plochy musí být vzhledem k počtu trojúhelníků, které obsahuje, velmi obtížené. VRUT ovšem obsahuje nástroj detekující, která hrana je součástí daného trojúhelníka a umí tak lehce říci zda stejnou hranu sdílí ještě s jiným trojúhelníkem. Proto nám po použití tohoto nástroje stačí projít všechny hrany a jejich počet sousedů. Pokud je hrana sdílena pouze jediným trojúhelníkem, tak nám automaticky vyplyne, že se jedná o okraj plochy.

7.1.1 Vzdálenost bodu a úsečky v prostoru

Pro výpočet vzdálenosti bodu a úsečky nemůžeme využít normálového vektoru, který bychom využili v rovině, neboť jak je známo úsečka má v prostoru nekonečné množství normálových vektorů. Z tohoto důvodu musí být využit jiný způsob výpočtu. Co tedy potřebujeme? Musíme nalézt bod, který leží na rovině. Rovina musí být kolmá k dané hraně a procházet originálním bodem hrany sousední plochy. Abychom toho dosáhli využijeme vlastnosti roviny, která je definována takto,

$$ax + by + cz + d = 0\,, \tag{30}$$

kde normálový vektor $\mathbf{n} = (a, b, c)$ je v našem případě směrový vektor úsečky. Dále potřebujeme nalézt průsečík roviny a přímky, která prochází body hrany $\mathbf{P}_1 = (x_1, y_1, z_1)$ a $\mathbf{P}_2 = (x_2, y_2, z_2)$. Spočteme tedy parametr u

$$u = \frac{ax_1 + by_1 + cz_1 + d}{a(x_1 - x_2) + b(y_1 - y_2) + c(z_1 - z_2)}\,. \tag{31}$$

Po dosazení do rovnice

$$\mathbf{P} = \mathbf{P}_1 + u(\mathbf{P}_2 - \mathbf{P}_1) \tag{32}$$

získáme požadovaný průsečík $\mathbf{P}$. Jestliže $u \in \langle 0, 1\rangle$ tak průsečík leží na úsečce $\mathbf{P}_1\mathbf{P}_2$ v opačném případě náleží přímce, která prochází body $\mathbf{P}_1\mathbf{P}_2$, ale není součástí úsečky. Tento problém je řešen v metodě `findOrInsertClosestVertex()`. Metoda funguje následujícím způsobem. Vstupem je bod úsečky datového typu `Vertex V`, který je součásti hrany jedné z ploch a úsečka zadaná krajními body reprezentující hranu druhé plochy $(\mathbf{P}_1, \mathbf{P}_2)$. Výstupem je informace uchována ve V, nesoucí informaci o sousedním bodu. Pokud je pointer v roven hodnotě `NULL`, pak v daném okolí nebyl nalezen vhodný sousední bod. V prvním kroku spočítáme vzdálenost mezi body $|\mathbf{PP}_1|$ a $|\mathbf{PP}_2|$. Pokud vzdálenost odpovídá toleranci, pak nastavíme potřebné parametry proměnné typu `Vertex` (proměnná `orig` nastavena na *true*) a danou metodu opouštíme. V opačném případě spočítáme vzdálenost bodu $\mathbf{P}$ a úsečky procházející bodem $\mathbf{P}_1$ a $\mathbf{P}_2$ a zkontrolujeme zda výsledný parametr $u \in \langle 0, 1\rangle$. Pokud ne, pak hledaný průsečík leží mimo úsečku a bod $\mathbf{P}$ nemá v dostatečné vzdálenosti sousední bod a daný proces je ukončen. V opačném případě nastavím proměnnou datového typu `Vertex` a končím. Tuto metodu volám nejdříve pro všechny okrajové body první plochy a poté pro okrajové body plochy druhé ovšem mimo ty, které už jsou svázané s protějšími originálními body. Po tomto procesu následuje roztřídění jednotlivých svázaných bodů (parametr `vert` není roven `NULL`) podle hranice, které náleží. Pro tento účel je využit námi definovaná struktura `Boundary`.

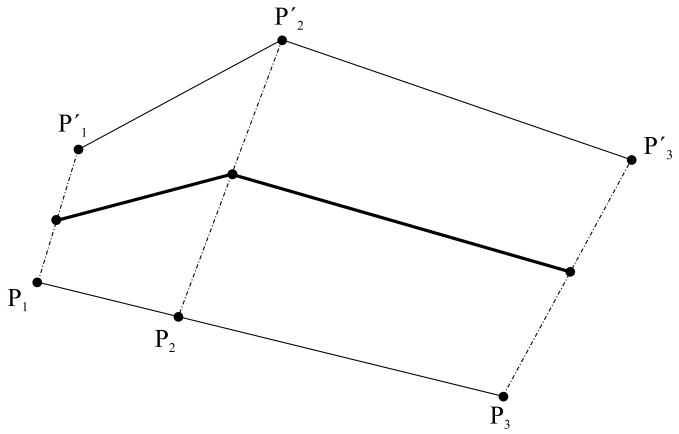
7.2 Reparametrizace a triangulace

Reparametrizace je proces, při kterém jsou nalezené sousední body posunuty tak, aby měly shodné souřadnice. Kromě reparametrizace je také potřeba u úseček, které obsahují

neoriginální body a které jsou rozděleny na dvě a více části, vytvořit nové trojúhelníky, pro které je ještě potřeba vypočíst nové normály. Proces reparametrizace se dělí na dvě části podle toho zda oba sousední body jsou originální či nikoliv. Pokud máme dva originální sousední body $\mathbf{P}_L$ a $\mathbf{P}_R$ z nichž každý náleží jiné ploše, pak nové hodnoty obou bodů získáme dosazením do následujícího jednoduchého vzorce

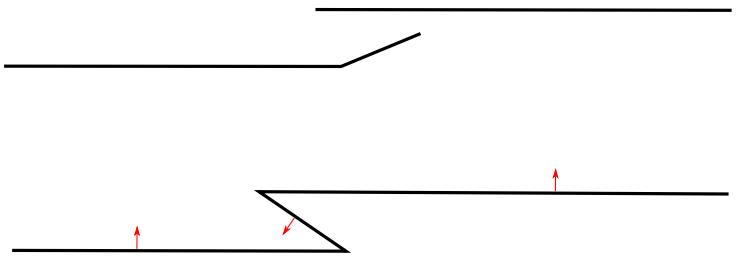
$$\mathbf{P} = \mathbf{P}_L + t(\mathbf{P}_R - \mathbf{P}_L), \quad \text{kde } t = 0,5$$
$$\mathbf{P}_L = \mathbf{P}_R = \mathbf{P}\,. \tag{33}$$

Dále je zřejmé, že zde nehrozí vznik nových trojúhelníků. Na závěr musím pro oba body přepočíst jejich normálové vektory.



Obrázek 35: Ukázka reparamterizace. Tlustá čára – reparametrizivaná hrana.

Daleko komplikovanější je reparametrizace u neoriginálního bodu, neboť při přepočtu nemůžeme vždy využít parametr o hodnotě 0.5. Důvodem proč to tak nejde ve všech případech jsou překryvy viz obrázek 36, které způsobují to, že normály vycházejí z daného bodu v opačném směru, což vede k nežádoucím úkazům.

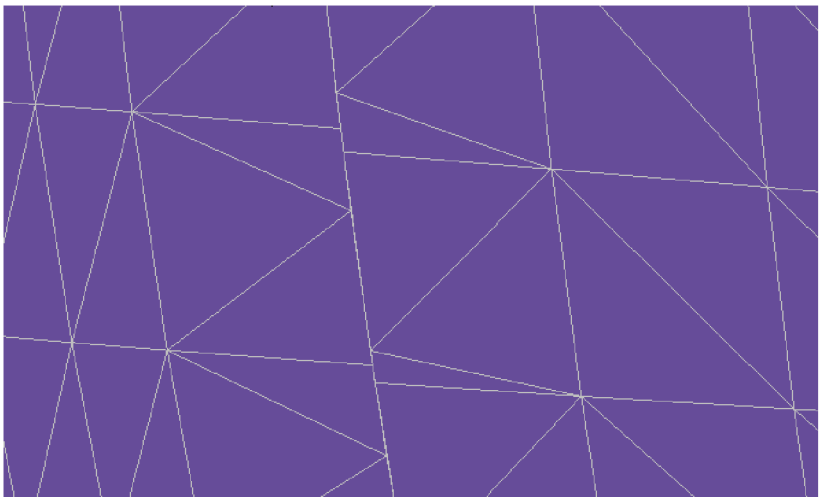


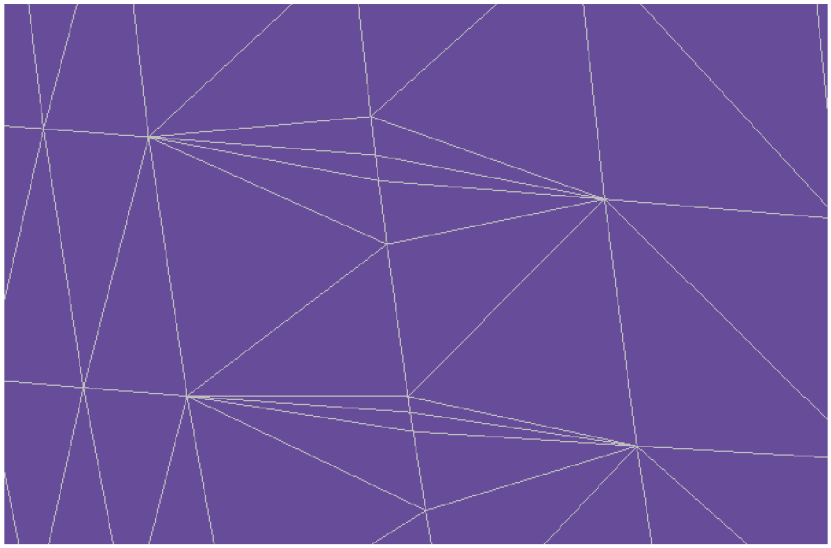
Obrázek 36: Ukázka problematiky překryvu z profilového pohledu

Touto reparametrizací se zabývá metoda `reparametrizeCreateNewTriangles()`. Cílem této metody je rozdělit původní trojúhelník, jehož hrana obsahuje neoriginální bod. Následující kroky naleznou požadovaný trojúhelník a potřebnou hranu. Tu posléze

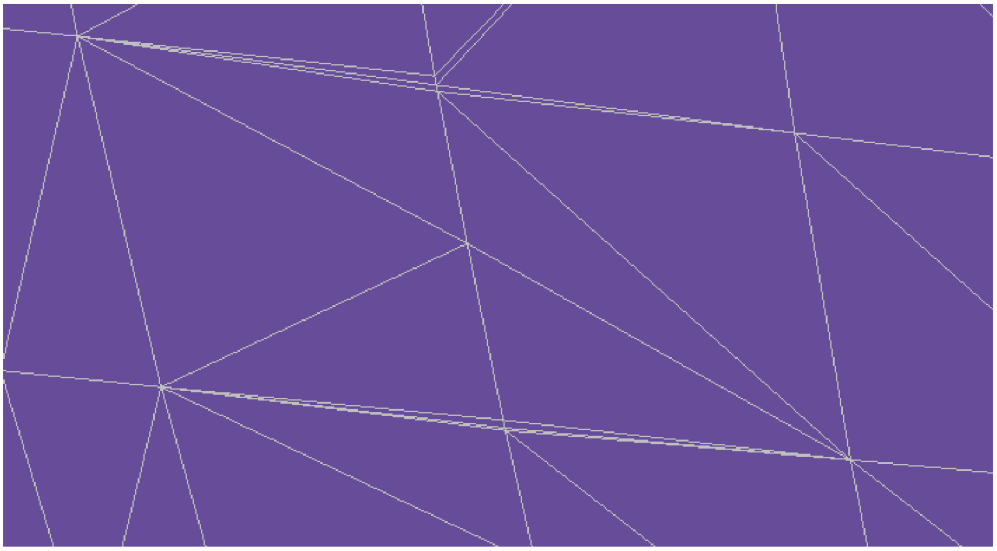
rozdělí podle počtu bodů, které se na ní nachází a následně jsou vytvořeny nové trojúhelníky. Důležitým krokem před samotným vytvářením je nový výpočet průsečíků, neboť při posuvu některého z předchozích bodů může dojít i k posunutí hrany. To může způsobit to, že i když provedu reparametrizaci daného bodu, tak stejně budu v překryvu jiných trojúhelníků. Po výpočtu průsečíku je tedy vytvořen nový trojúhelník, který je ze začátku reparametrizován hodnotou 0.5 a poté otestován zda se nejedná o překryv. Problematiku překryvů řeší metoda `findOverlap()`. Vstupem je nově vytvořený trojúhelník, výstupem pak informace o tom zda leží v překryvu či nikoliv. Prvním krokem tohoto algoritmu je nalezení všech trojúhelníků, které se vstupním trojúhelníkem sousedí. Pokud při dělení hrany v předchozím kroku vzniká více trojúhelníků, tak ty co by po vstupním trojúhelníku následovaly ještě nejsou vytvořeny a teoreticky tak jedné hraně soused chybí. Abych tento problém vyřešil musel jsem vytvořit dočasný trojúhelník, jehož hrana začínala stejným bodem jako má vstupní trojúhelník a končila krajním bodem dělené hrany. Po nalezení všech sousedů vypočtu každému z nich normálový vektor a poté spočítám ku vstupnímu trojúhelníku úhel, který spolu tyto dva normálové vektory svírají. Pokud vypočtený úhel vždy odpovídá intervalu $\langle 0, 90 \rangle$ pak vracím hodnotu *true* a končím algoritmus v opačném případě vracím hodnotu *false*.

V případě, že mi metoda `findOverlap()` vrátí hodnotu *false*, pak jeden bod reparametrizuji hodnotou 0 a druhý hodnotou 1 a znovu provedu test na překryv. Pokud opět obdržím hodnotu *false* tak hodnoty parametrů prohodím ovšem musím pracovat s původními hodnotami, které mám zálohované. Je zřejmé, že v případě, kdy hned na začátku obdržím hodnotu *true* jsou zde uvedené kroky přeskočeny. Na závěr už jen stačí pro každý vrchol nově vytvořených trojúhelníků spočítat normálové vektory. Stejnou operaci je potřeba provést i pro reparametrizované originální body. Dále bylo třeba celou dobu dodržovat řazení bodů proti směru hodinových ručiček, neboť v opačném případě by vycházely normálové vektory opačného směru. Následující obrázky 37 ukazují stav před použitím „Sewing“ algoritmu a výsledné sešití.





Obrázek 37: Ukázka dvou ploch před (horní) a po sešití (dolní)



Obrázek 38: Ukázka trojúhelníku s ilegální hranou

Jak je vidět z předchozíhc obrázků 38 37, nevýhodou tohoto postupu jsou ilegální hrany na nově vzniklých trojúhelnících. Důvod proč tato situace nastává, je rychlost takovéhoto sešití, neboť by jsme museli vyhledávat vhodné trojúhelníky v celé jejich množině.

8 Závěr

V rámci mé diplomové práce byly vytvořeny dva moduly pro nástroj virtuální reality VRUT. První dokáže zpracovat soubor formátu IGES a uložit do jádra VRUTU vytipované entity, obsahující především NURBS plochy a křivky. Co se týče rychlosti zpracování, tak mnou vytvořený modul zpracovává soubory přibližně stejnou v jistých případech i vyšší rychlostí jako komerční nástroje. Pro přesnost jen uvedu, že výsledný odhad probíhal na základě logovacích výpisů jednotlivých programů a nebylo tak využito komerčních měřících nástrojů. Jistou nedokonalostí tohoto modulu je prozatimní absence zpracování, některých méně používaných entit, které budou dle potřeb později dodělávány.

Druhý modul byl vytvořen pro potřebu tesselace jakékoliv ořezané NURBS plochy. Součástí práce bylo objevit a vyzkoušet co nejoptimálnější algoritmy, aby se daný modul přibližoval svými výsledky, co se týče počtu trojúhelníků a rychlostí, používaným komerčním nástrojům. Výsledný modul je rychlostně i kvalitativně použitelný co se týče samotného nasazení, kde i sešívací algoritmus odpovídá požadavkům a lze ho považovat za srovnatelný s ostatními produkty. Co se ovšem nepovedlo je nedosažení naprosto srovnatelných výsledků s komerčními programy. Díky rozdílnému použití aproximační metody pro plochy je rozložení a samotný počet trojúhelníků zatím ku neprospěchu mého modulu. Lepších výsledků by bylo možné dosáhnout, pomocích druhých derivací a lepšího analytického odhadu křivosti křivky. Důvodem proč nebyla aplikována tato metoda je fakt, že nebyl nalezen dostatečný materiál, věnující se této problematice. Naopak srovnatelných kvalit dosahuje použitá triangulace a aproximace ořezových křivek. Dalšími problémy, které nebyly dosud vyřešeny jsou občasné problémy při zaokrouhlování, jenž způsobují chyby především u algoritmu lokalizujícího bod v polygonu a také u determinantu, který testuje ilegálnost nově vložených hran. První chyba se projevuje nadbytečnými obdélníky a tím i trojúhelníky ve výsledné scéně. Druhá chyba může být pozorovatelná, především na výsledném tvaru triangulace. Poslední slabinou celého programu je zatím pomalý Sewing algoritmus, který je způsobem neoptimálním vyhledávání sousedních ploch, jenž má zatím pouze kvadratickou složitost.

9 Reference

[1] U.S. Product Data Association. *Initial Graphics Exchange Specification 5.3.* Formerly ANS US PRO/IPO-100-1996.

[2] MÍŠEK Antonín, KYBA Václav. Škoda-auto a.s. ČVUT v Praze. *Dokumentace aplikace VRUT.* 2011.

[3] SLABÁKOVÁ, Jana. *Modelování NURBS křivek a ploch v projektivním prostoru.* Brno, 2009. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce doc. PaedDr. Dalibor Martišek, Ph.D.

[4] SOJKA Eduard, FABIÁN Tomáš a NĚMEC Martin. *Matematické základy počítačové grafikyl.* 2011.

[5] PIEGL, Les. *The NURBS book.* 2nd ed. Berlin: Springer-Verlag, 1997, 300 s. ISBN 35-406-1545-8.

[6] KAHLESZ F., BALÁSZ Á. a KLEIN R. *Multiresolution rendering by sewing trimmed NURBS surfaces.* In Proceedings of Symposium on Solid Modeling and Applications. 2002, 281-288.

[7] WEILER, Kevin a Peter ATHERTON. *Hidden Surface Removal Using Polygon Area Sorting.* Computer Graphics, 1977.

[8] ANGLADA, M. V. *An improved incremental algorithm for constructing Restricted Delaunay triangulation.*, Comput. and Graphics, 21st edition, 1997. s. 215 – 223.

[9] MIROSLAV, Fuksa. *Delaunayova triangulace s omezením (CDT) v E2 a E3.* Plzeň, 2006. Diplomová práce. Západočeská univerzita v Plzni.

[10] KOHOUT, Josef. *Paralelní Delaunayova triangulace ve 2D a 3D.* Plzeň, 2002. Diplomová práce. Západočeská univerzita v Plzni.

[11] MACRI, Dean. INTEL CORPORATION. *Using NURBS Surfaces in Real-time Applications.* 2000. Dostupné z: http://software.intel.com/file/27081.

[12] KRISHNAMURTHY, Adarsh, Rahul KHARDEKAR, Sara MCMAINS, Kirk HALLER a Elber GERSHON. *Performing Efficient NURBS Modeling Operations on the GPU.* IEEE Transactions on Visualization and Computer Graphics, vol. 15, no. 4, pp. 530-543, July-Aug. 2009, doi:10.1109/TVCG.2009.29.

[13] MALLÓN N. Paula, BÓO Montserrat a Javier D. BRUGUERA. *Implementation of a NURBS to Bézier Conversor with Constant Latency.* Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-42499-4. Dostupné z: http://dx.doi.org/10.1007/3-540-44687-722.

[14] JAHODA, Mirek. *Co přináší teselace + první DirectX 11 hry.* Extrahardware.cz. Dostupné z: http://extrahardware.cnews.cz/co-prinasi-teselace-prvni-directx-11-hry. [Online] [Datum: 22.04.2012]

[15] SURYNKOVÁ, Petra. *Počítačová geometrie.* Dostupné z: http://www.surynkova.info/dokumenty/mff/ PG/ Prednasky/prednaska11.pdf. [Online] [Datum: 20.04.2012]

[16] *Bézier Surfaces: de Casteljau's Algorithm.* Dostupné z: http://www.cs.mtu.edu/shene/COURSES/cs3621/NOTES/surface/bezier-de-casteljau.html. [Online] [Datum: 20.4.2012]

[17] *De Boor's Algorithm.* Dostupné z: http://www.cs.mtu.edu/ shene/ COURSES/cs3621 / NOTES/spline/ B-spline/de-Boor.html. [Online] [Datum: 20.04.2012]

[18] BAYER, Tomáš. *Voronoi diagram: Vlastnosti, použití, konstrukce. Zobecnené Voronoi diagramy.* Dostupné z: http://web.natur.cuni.cz/ bayertom/Adk/adk6.pdf. [Online] [Datum: 26.4.2012]

[19] *Bézier surface.* In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: http://en.wikipedia.org/wiki/Bezier_surface. [Online] [Datum: 27.4.2012]

A Ukázka IGES souboru

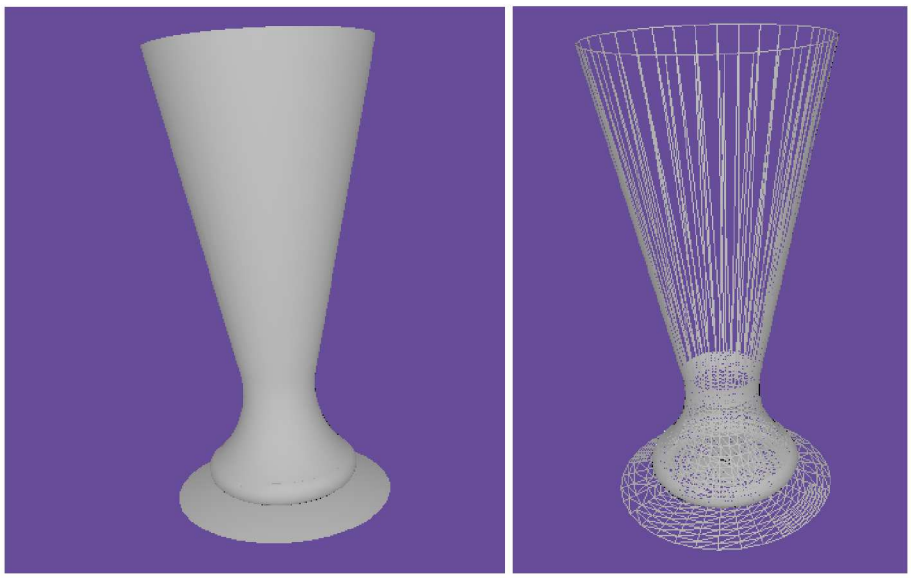
Ukázka vstupního IGES souboru, který by měl reprezentovat obdélníkovou plochu.

```
START RECORD GO HERE.                                                    S      1
1H,,1H;,5HPART9,12HOBDELNIK.igs,44HDASSAULT SYSTEMES CATIA V5 R19 - www.G 1
3ds.com,32HCATIA Version 5 Release 19 SP 9 ,32,75,6,75,15,5HPART9,1.0,2,G 2
2HMM,1000,1.0,15H20120323.155914,0.001,10000.0,7HDZCIU03,8HCZSKDA01,11, G 3
0,15H20120323.155914,;                                                   G      4
     126       1       0       0       0       0       0       001010001D      1
     126       0       0       2       0                               0D      2
     126       3       0       0       0       0       0       001010001D      3
     126       0       0       2       0                               0D      4
     126       5       0       0       0       0       0       001010001D      5
     126       0       0       2       0                               0D      6
     126       7       0       0       0       0       0       001010001D      7
     126       0       0       2       0                               0D      8
     102       9       0       0   10000       0       0       000000000D      9
     102       0       0       1       0                Sketch.1       0D     10
     128      10       0       0       0       0       0       001010001D     11
     128       0       0       3       0                               0D     12
     126      13       0       0       0       0       0       001010001D     13
     126       0       0       2       0                               0D     14
     126      15       0       0       0       0       0       001010001D     15
     126       0       0       2       0                               0D     16
     126      17       0       0       0       0       0       001010001D     17
     126       0       0       2       0                               0D     18
     126      19       0       0       0       0       0       001010001D     19
     126       0       0       2       0                               0D     20
     102      21       0       0       0       0       0       001010001D     21
     102       0       0       1       0                               0D     22
     126      22       0       0       0       0       0       001010501D     23
     126       0       0       2       0                               0D     24
     126      24       0       0       0       0       0       001010501D     25
     126       0       0       2       0                               0D     26
     126      26       0       0       0       0       0       001010501D     27
     126       0       0       2       0                               0D     28
     126      28       0       0       0       0       0       001010501D     29
     126       0       0       2       0                               0D     30
     102      30       0       0       0       0       0       001010501D     31
     102       0       0       1       0                               0D     32
     142      31       0       0       0       0       0       001010001D     33
     142       0       0       1       0                               0D     34
     144      32       0       0   10000       0       0       000000000D     35
     144       0       0       1       0                 Fill .1       0D     36
126,1,1,0,0,1,0,-35.0,-35.0,35.0,35.0,1.0,1.0,-100.0,40.0,0.0,          1P      1
-100.0,-30.0,0.0,-35.0,35.0,0.0,0.0,0.0,0.0,0;                          1P      2
126,1,1,0,0,1,0,0.0,0.0,180.0,180.0,1.0,1.0,-100.0,-30.0,0.0,           3P      3
 80.0,-30.0,0.0,0.0,180.0,0.0,0.0,0.0,0.0,0;                            3P      4
126,1,1,0,0,1,0,-35.0,-35.0,35.0,35.0,1.0,1.0,80.0,-30.0,0.0,           5P      5
 80.0,40.0,0.0,-35.0,35.0,0.0,0.0,0.0,0.0,0;                            5P      6
 126,1,1,0,0,1,0,0.0,0.0,180.0,180.0,1.0,1.0,80.0,40.0,0.0,             7P      7
```

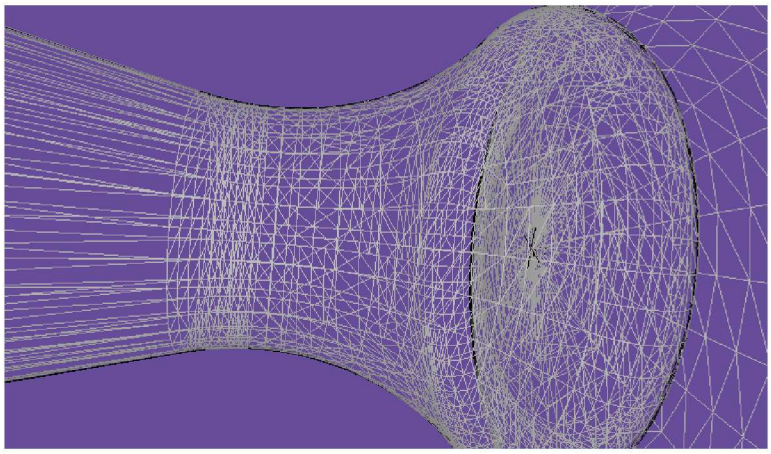
−100.0,40.0,0.0,0.0,180.0,0.0,0.0,0.0,0.0,0;	7P	8
102,4,1,3,5,7,0,0;	9P	9
128,1,1,1,1,0,0,1,0,0,−100.0,−100.0,80.0,80.0,−30.0,−30.0,40.0,	11P	10
40.0,1.0,1.0,1.0,1.0,−100.0,−30.0,0.0,80.0,−30.0,0.0,−100.0,	11P	11
40.0,0.0,80.0,40.0,0.0,−100.0,80.0,−30.0,40.0,0,0;	11P	12
126,1,1,0,0,1,0,−35.0,−35.0,35.0,35.0,1.0,1.0,−100.0,40.0,0.0,	13P	13
−100.0,−30.0,0.0,−35.0,35.0,0.0,0.0,0.0,0.0,0;	13P	14
126,1,1,0,0,1,0,0.0,0.0,180.0,180.0,1.0,1.0,−100.0,−30.0,0.0,	15P	15
80.0,−30.0,0.0,0.0,180.0,0.0,0.0,0.0,0.0,0;	15P	16
126,1,1,0,0,1,0,−35.0,−35.0,35.0,35.0,1.0,1.0,80.0,−30.0,0.0,	17P	17
80.0,40.0,0.0,−35.0,35.0,0.0,0.0,0.0,0.0,0;	17P	18
126,1,1,0,0,1,0,0.0,0.0,180.0,180.0,1.0,1.0,80.0,40.0,0.0,	19P	19
−100.0,40.0,0.0,0.0,180.0,0.0,0.0,0.0,0.0,0;	19P	20
102,4,13,15,17,19,0,0;	21P	21
126,1,1,1,0,1,0,−35.0,−35.0,35.0,35.0,1.0,1.0,−100.0,40.0,0.0,	23P	22
−100.0,−30.0,0.0,−35.0,35.0,0.0,0.0,0.0,1.0,0,0;	23P	23
126,1,1,1,0,1,0,0.0,0.0,180.0,180.0,1.0,1.0,−100.0,−30.0,0.0,	25P	24
80.0,−30.0,0.0,0.0,180.0,0.0,0.0,0.0,1.0,0,0;	25P	25
126,1,1,1,0,1,0,−30.0,−30.0,40.0,40.0,1.0,1.0,80.0,−30.0,0.0,	27P	26
80.0,40.0,0.0,−30.0,40.0,0.0,0.0,0.0,1.0,0,0;	27P	27
126,1,1,1,0,1,0,0.0,0.0,180.0,180.0,1.0,1.0,80.0,40.0,0.0,	29P	28
−100.0,40.0,0.0,0.0,180.0,0.0,0.0,0.0,1.0,0,0;	29P	29
102,4,23,25,27,29,0,0;	31P	30
142,0,11,31,21,1,0,0;	33P	31
144,11,1,0,33,0,0;	35P	32
S 1G 4D 36P 32	T	1

B Výsledky

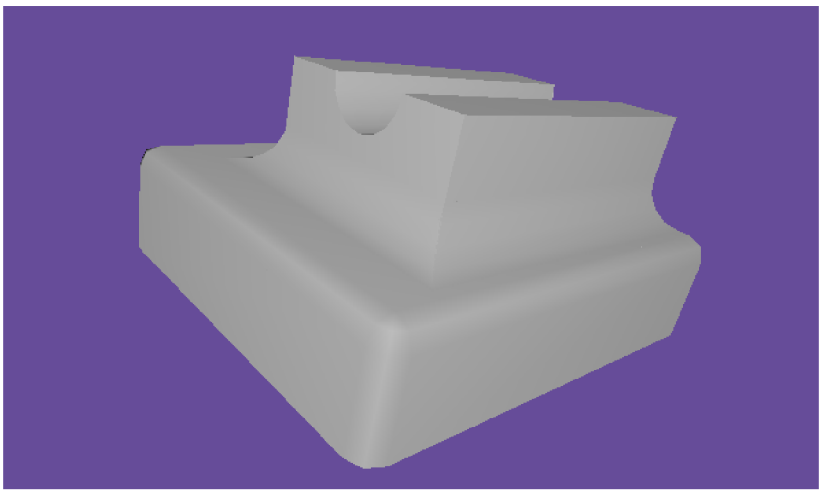
Ukázka výsledků tesselace mého tesselátoru.



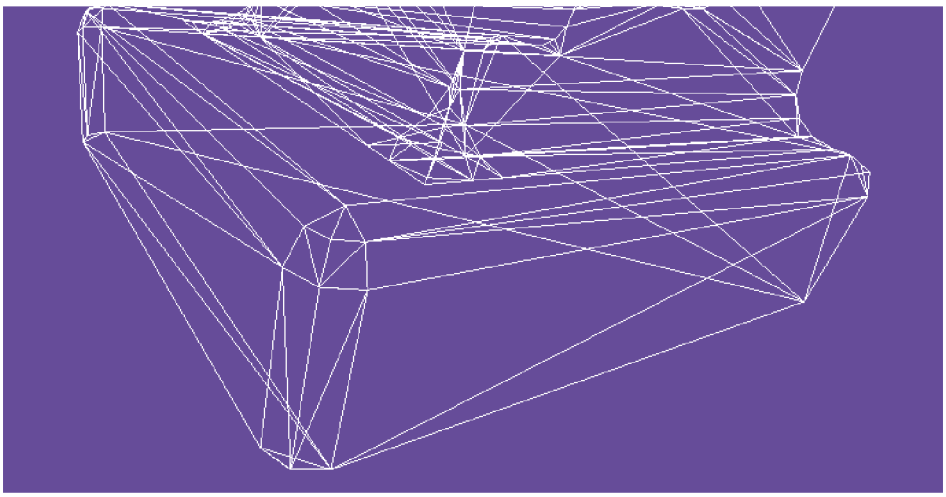
Obrázek 39: Výsledek



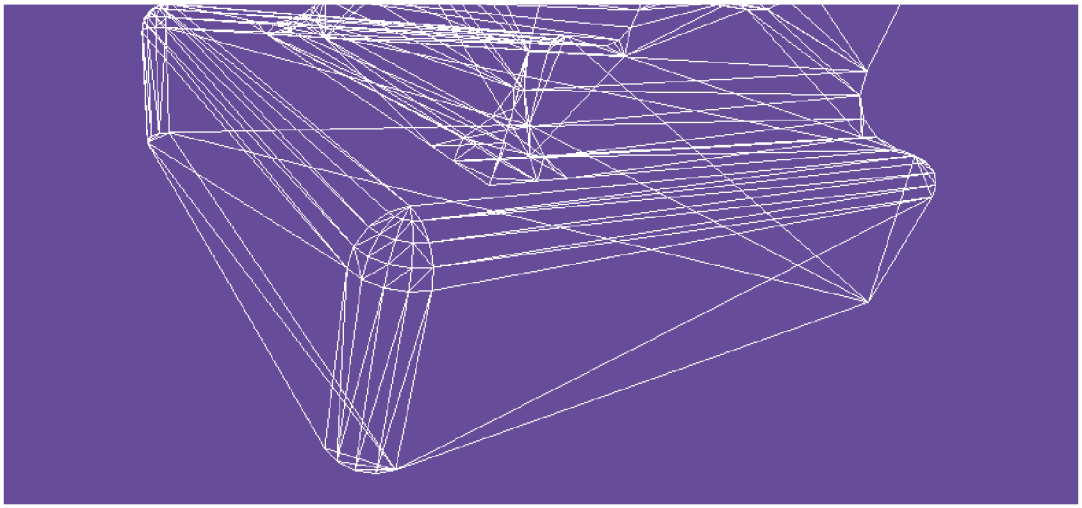
Obrázek 40: Výsledek



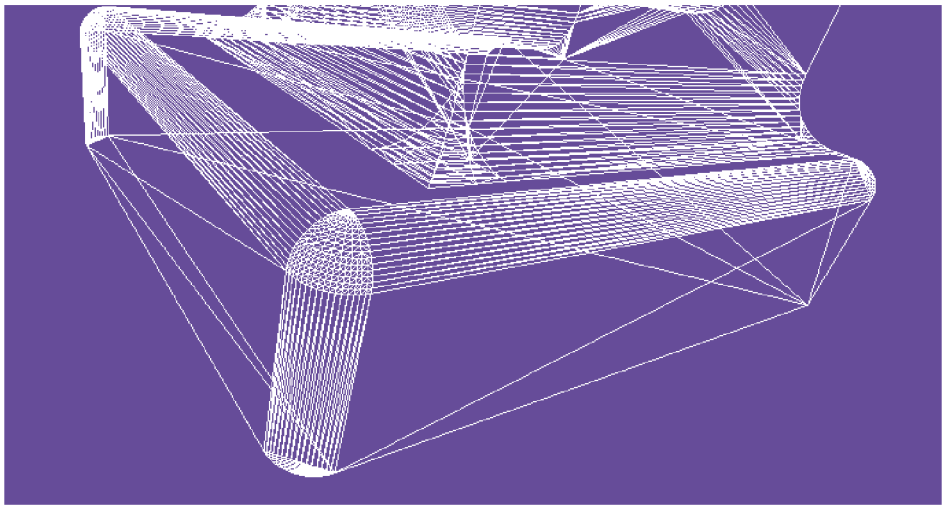
Obrázek 41: Výsledek



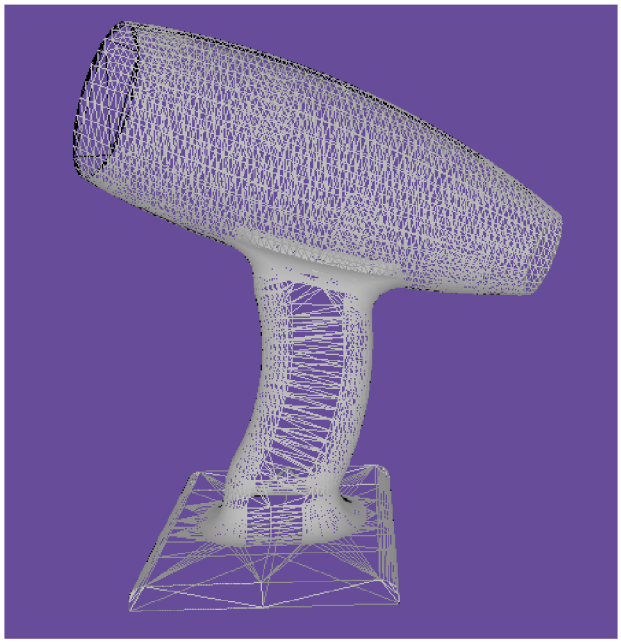
Obrázek 42: Výsledek při toleranci 0.2



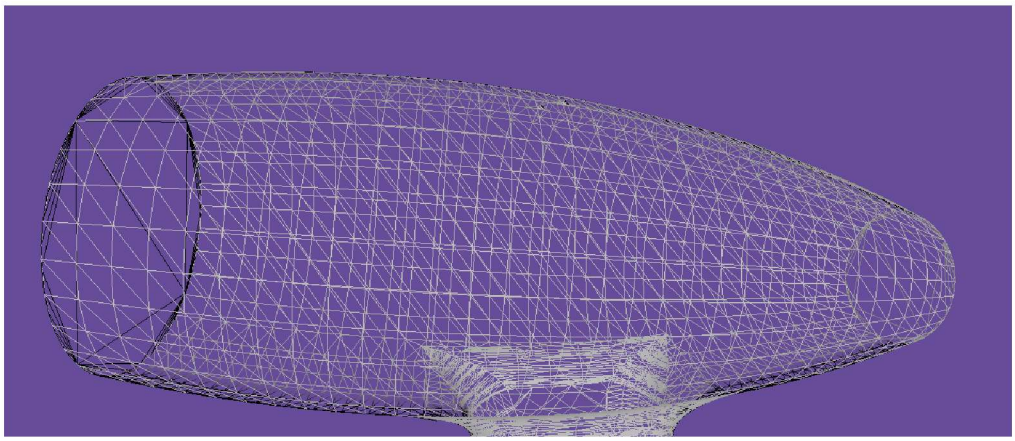
Obrázek 43: Výsledek při toleranci 0.1



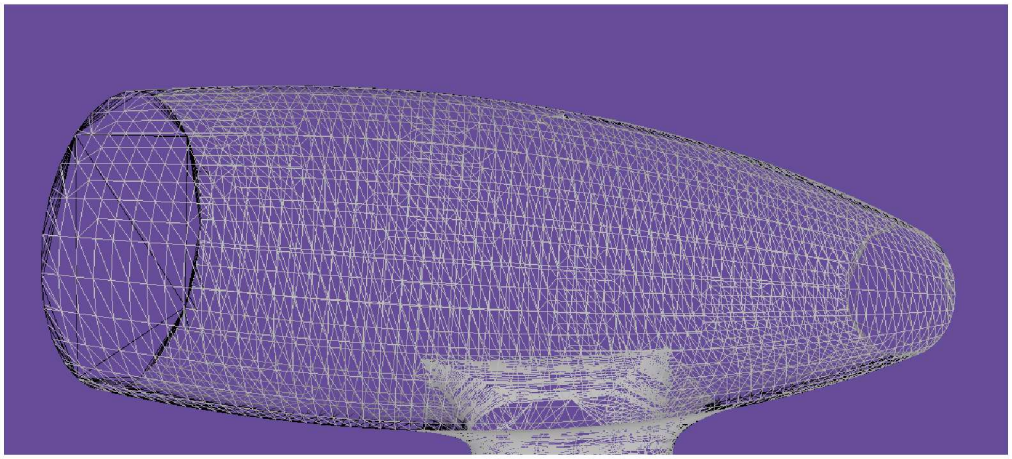
Obrázek 44: Výsledek při toleranci 0.02



Obrázek 45: Výsledek při toleranci 0.1



Obrázek 46: Výsledek při toleranci 0.2



Obrázek 47: Výsledek při toleranci 0.1